Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Clasificación de 3D Point Cloud mediante descriptores VFH

Autor: Sara García Navarro

Tutor/es: Rafael Barea Navarro

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería electrónica y automática industrial

Trabajo Fin de Grado

Clasificación de 3D Point Cloud mediante descriptores VFH

Autor: Sara García Navarro

Tutor/es: Rafael Barea Navarro

TRIBUNAL:

Presidente: LUIS MIGUEL BERGASA PASCUAL

Vocal 1º: MIGUEL ÁNGEL GARCÍA GARRIDO

Vocal 2º: RAFAEL BAREA NAVARRO

FECHA:

AGRADECIMIENTOS

Gracias a mi familia y amigos por el cariño, la ayuda y apoyo incondicional durante todos estos años. A todos los que habéis formado parte de mi crecimiento personal y académico, que siempre me impulsaron a seguir hacia delante.

Gracias a mi tutor Rafael Barea por esta gran oportunidad, y sobre todo por su atención, paciencia y apoyo en estos últimos meses. También quiero agradecer a todos mis profesores y compañeros que me brindaron la oportunidad de aprender a su lado.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN EN CASTELLANO

El objetivo principal de este proyecto es realizar el reconocimiento de objetos mediante el uso de descriptores VFH (Viewpoint Feature Histogram), que permiten clasificar objetos a partir de similitudes geométricas.

Con el entorno de simulación V-REP, se generarán una base de datos y unos escenarios con diferentes objetos. Serán captados por los sensores de un vehículo automatizado, obteniendo una nube de datos 3D. Se empleará la herramienta PCL (Point Cloud Library), que cuenta con módulos que permiten trabajar con este tipo de datos y con el descriptor VFH, de forma que se puedan clasificar los obstáculos.

Palabras Clave: cluster, nube de puntos, LIDAR, V-REP, VFH.

ABSTRACT

The main objective of this project is the recognition of objects through the use of Viewpoint Feature Histogram, which allows to classify objects from geometric similarities.

Using the simulation tool V-REP, you can obtain a data base and scenes with different obstacles. They will be captured by the sensors of an automated vehicle, obtaining a 3D point cloud. The Point Cloud Library will be used, which has modules that allow working with this type of data and VFH, so that obstacles can be recognized.

Keywords: cluster, point cloud, LIDAR, V-REP, VFH.

RESUMEN EXTENDIDO

La automatización es una tecnología cuyo objetivo es el empleo de máquinas y sistemas de control para tareas sin necesidad de la intervención del ser humano. Gracias a ella, se consigue reducir el tiempo y el coste de muchos procesos.

En la actualidad, las empresas automovilísticas están desarrollando distintas tecnologías para automatizar los vehículos. Para ello se combina la mecánica, electrónica, los distintos sistemas de visión e inteligencia artificial, junto con los sistemas de navegación para conseguir la conducción autónoma.

La conducción autónoma debe realizarse de forma exacta y con una tecnología que sea capaz de reaccionar ante diversos escenarios de forma más eficaz con la que lo haría el ser humano. Para ello, el sistema deberá ser capaz de situarse en el entorno y reconocer cada uno de los obstáculos que se le presenten.

Actualmente, los sistemas de navegación GPS incluyen información de las carreteras, como: posibles rutas origen-destino, limitaciones de velocidad, ubicación de señales de tráfico, etc., que pueden servir como guía a la hora de conducir, adaptando el coche a la velocidad adecuada.

Pero esta información solo sería útil a la hora de realizar el control del vehículo automatizado si no existieran más vehículos en la carretera. Existen numerosos obstáculos, como pueden ser otros vehículos, semáforos que permiten el acceso a la vía o no de manera cambiante, peatones, obstáculos en medio de la carretera (durante unas obras puede haber señales, operarios, conos) que se deberán tener en cuenta durante la conducción.

Para el reconocimiento del resto de obstáculos, se emplea la **visión artificial**, ciencia que se encarga de adquirir, procesar y analizar las imágenes del medio de forma que se puedan tratar de forma computacional.

Mediante diferentes sensores, como cámaras o escáneres laser, se capturan los datos del exterior y se procesan de forma que se pueda reconocer qué objetos hay en el entorno.

En este *trabajo de fin de grado*, se reconocerán los diferentes obstáculos posibles que un vehículo se puede encontrar durante la conducción.

Para ello, se empleará un escáner laser **LIDAR** que mediante haces de rayos es capaz de detectar la presencia de un obstáculo y saber a qué distancia se encuentra este, de forma que se elabora un modelo en 3D del entorno. La empresa **Velodyne** se encarga de construir LIDAR que se puedan emplear cuando un vehículo se encuentra en movimiento.

Otra forma de detectar objetos es mediante **imágenes de profundidad**: colocando al menos una cámara en cada extremo del vehículo, se puede formar una imagen de profundidad de la que se obtiene un modelo geométrico en 3D.

Estos modelos en 3D del entorno, se procesan como **nubes de puntos**, pudiendo contener además de las coordenadas XYZ de cada punto, información del color e intensidad de dichos puntos.

El hecho de contener información de color hace que la nube de puntos se pueda segmentar por objetos en función de los colores: bien sea eliminando los objetos de algún color que no nos interese u obteniendo los objetos de los colores que nos interesen.

El método empleado para el reconocimiento de objetos es mediante el uso de descriptores, que mediante la estimación de las normales de los puntos que forman la nube, se puede obtener la forma geométrica del objeto. Esta forma geométrica se compara con una base de datos, que está formada por los diferentes objetos que se quieren identificar, almacenados en distintas posiciones.

Para ello, se creará una base de datos para cada método de adquisición de nubes de puntos 3D. Cada base de datos estará formada por los principales obstáculos que preocupan a un conductor: los vehículos y los peatones, ya que durante la conducción habrá que tener en cuenta su trayectoria y comportamiento para no colisionar con ellos.

La herramienta PCL dispone de módulos para trabajar tanto con nubes de puntos 3D como con el descriptor VFH.

1 INTRODUCCIÓN

1.1 ESTADO DEL ARTE

1.1.1 Reconocimiento de objetos mediante descriptores

Uno de los métodos empleados para el reconocimiento de objetos 3D es mediante el uso de **descriptores geométricos**. Los descriptores permiten el reconocimiento a partir de la semejanza entre 2 objetos, normalmente empleando bases de datos. Los objetos 3D se representan como una nube de puntos, de las que se extraerán las características geométricas de la superficie. Se diferencian tres grupos en los que se pueden clasificar los descriptores geométricos:

- Características de los puntos de las nubes y de los puntos vecinos (PFH, FPFH).
- Características del punto de vista (**VFH**, CVFH).
- Orientación de su superficie (SHOT).

En este trabajo de fin de grado se estudiará el descriptor VFH que combina las características geométricas de la nube de puntos y sus vecinos añadiendo las características del punto de vista.

1.1.2 Proyectos previos

En este apartado, se comentarán algunos proyectos que realizan el reconocimiento de objetos empleando diferentes tipos de descriptores.

Proyecto IROS *(Intelligents RObots and System):* Willow Garage está dedicado al diseño de software de código abierto en el ámbito de la robótica y visión artificial. Presentan una alternativa al reconocimiento de objetos, mediante el uso de descriptores. Entre ellos, se encuentra el descriptor VFH [1], que es capaz de realizar el reconocimiento de objetos y la estimación de la posición mediante comparaciones geométricas. Obtienen un porcentaje del 98,52% de probabilidad de acierto tanto en reconocimiento de objetos como de estimación de la pose, empleando 54000 escenas diferentes con más de 60 objetos.

Point Cloud Library: Esta herramienta [2] está provista de módulos que trabajan con los descriptores para el reconocimiento de objetos, también creada por el laboratorio de investigación Willow Garage, de forma que se puedan utilizar de forma sencilla.

MuCAR-3: Munich Cognitive Autonomous Robot Car 3rd Generation [3], es un proyecto llevado a cabo por la universidad de Bundeswehr Muniches. Realizan la detección de objetos en tiempo real empleando un LIDAR con el descriptor PFH (base del VFH), obteniendo un 96,7% de aciertos.

Universidad de León: el grupo de robótica de la Universidad de León [4] está dedicado al estudio del comportamiento de robots autónomos. En el proyecto *PhD-3D-Object-Tracking* realizado en 2011, recopilan mediante una serie de tutoriales, diferentes estudios del reconocimiento y seguimiento de objetos, a partir del procesamiento de nubes de puntos, empleando la herramienta PCL y los diferentes tipos de descriptores para la identificación de los objetos.

1.2 OBJETIVOS DEL PROYECTO

El objetivo de este trabajo de fin de grado es, mediante el empleo de descriptores VFH, reconocer los diferentes objetos que un vehículo automatizado se encuentre durante la conducción.

En primer lugar, se creará una base de datos mediante el simulador V-REP con los principales obstáculos que preocupan al conductor: los vehículos y los peatones. En función de su comportamiento, el conductor o el vehículo automatizado deberá tomar decisiones de forma que no se ponga en peligro ni al conductor ni al resto de personas que circulan. Es por ello que se deberá realizar un seguimiento y reconocimiento de los diferentes obstáculos de forma correcta.

En segundo lugar, se generarán diferentes escenarios con los obstáculos tanto detenidos como en movimiento. El vehículo adquirirá los datos del exterior mediante dos métodos: uno de menor resolución de datos a mayor distancia al objetivo, y otro de mayor resolución de datos, pero a menor distancia. Con la combinación de ambos, se podrá reconocer los obstáculos mediante comparaciones geométricas con la base de datos. El esquema quedaría de la siguiente forma:

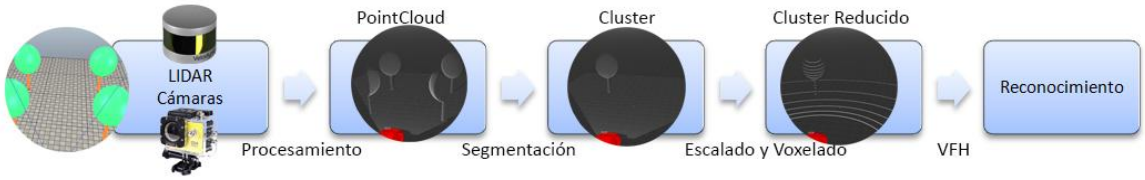


Ilustración 1-1: Esquema del reconocimiento de objetos

1.3 ARQUITECTURA DEL PROYECTO

En este apartado, se detallará la arquitectura que se seguirá para la realización de este trabajo de fin de grado:

- **Arquitectura y Software:** en este apartado, se detallará la estructura del sistema que se empleará para obtener los datos del entorno, así como el software necesario para procesarlos. Con el simulador V-REP, se crearán diferentes escenarios en 3D, formados por los diferentes obstáculos a segmentar y reconocer.
- **Diseño de la aplicación:** una vez creados los modelos 3D, se procede a diseñar la aplicación, que constará de varios pasos:
    - *Base de datos:* se creará una base de datos para cada método de adquisición de datos. Cada obstáculo se extraerá de una nube de puntos, se rotará sobre su eje Z para obtener diferentes posiciones y así realizar un reconocimiento más preciso.
    - *Escalado y voxelado:* todos los objetos de la base de datos se filtran para reducir el ruido y se escalan, de forma que todos tengan en común la longitud del eje Z.
    - *Extracción de características VFH:* formada la base de datos, se calcula el descriptor de cada objeto, formando un histograma que contiene información de la forma geométrica de cada objeto y de su punto de vista.

- **Simulación y resultados:** con el simulador V-REP, se crearán diferentes casos de uso (obstáculos en movimiento, detenidos, con diferentes trayectorias), y se empleará el algoritmo diseñado para realizar el seguimiento y reconocimiento de los objetos para ambos casos.

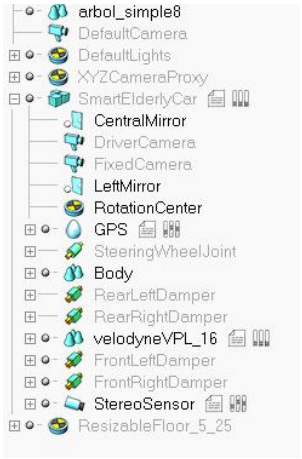
2 ARQUITECTURA Y SOFTWARE

2.1 COCHE SMART ELDERLY CAR

Es un proyecto realizado por la Universidad de Alcalá y la Universidad de Vigo, cuyo objetivo es desarrollar un vehículo automatizado capaz de moverse en entornos urbanos.

Para ello, se diseñará un vehículo formado por distintos tipos de sensores: láser, GPS, y un sensor estéreo, consiguiendo así mapear, segmentar la escena e identificar los objetos presentes en ella.

En el entorno de simulación V-REP utilizado en este trabajo, se creará un modelo de dicho vehículo. Este modelo estará formado por el chasis y la carrocería (para simular el aspecto de un vehículo real), y tres modelos que simulan los sensores que componen el vehículo sensorizado: *velodyneVPL_16*, *StereoSensor*, y *GPS.* Los modelos de los sensores están programados mediante unos *scripts* que definen su comportamiento, de forma que sean como el de los sensores reales. La estructura del modelo es la siguiente:



Además del *script*, los sensores tienen unos parámetros de resolución y ruido que hacen que sea más parecido al modelo real:

Sensor	Parámetro	Valor
StereoSensor	*sensorResolutionX*	800 px
	sensorResolutionY	800 px
	interOcularDistance	0.07 m
	focusAngle	0 º
	sensorAperture	80 º
	sensorFarClippingPlane	300 m
	sensorNearClippingPlane	0.02 m

GPS	*xNoiseAmplitude*	0.1 m
	yNoiseAmplitude	0.1 m
	zNoiseAmplitude	0.05 m
	xShiftAmplitude	1 m
	yShiftAmplitude	1 m
	zShiftAmplitude	0.5 m

Tabla 1: Parámetros de los sensores del vehículo

El modelo resultante queda de la siguiente forma en el simulador de V-REP, con los sensores situados en la parte superior:

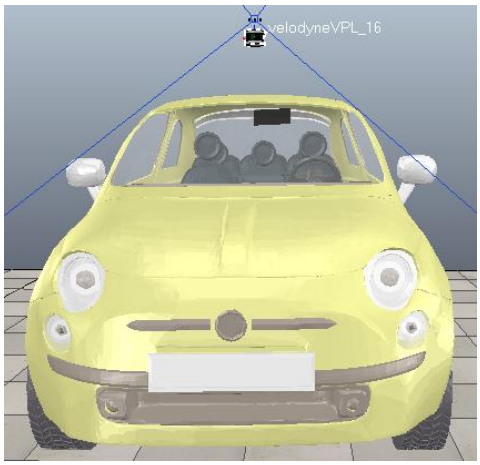


Ilustración 2-1: Modelo del vehículo en V-REP

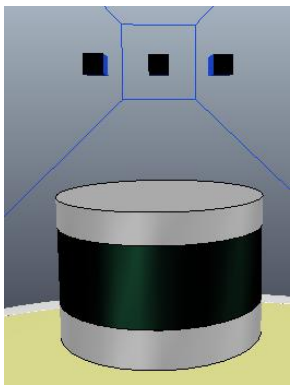


Ilustración 2-2: Modelo de los sensores del vehículo en V-REP

El LIDAR se encuentra a una altura de 2 metros, y el sensor estéreo a una altura de 2,1 metros. El sensor estéreo está formado por 2 sensores: uno a la izquierda y otro a la derecha que capturarán las imágenes del entorno, separados por una distancia de 1,2 mm. Con estos dos sensores estéreos se creará una imagen 3D del entorno.

2.1.1 VLP-16

Lídar: (un acrónimo del inglés **LIDAR**, ***LI**ght **D**etection **A**nd **R**anging* o ***L**aser **I**maging **D**etection **A**nd **R**anging*, detección por luz y distancia*).* Es una técnica de detección óptica que emplea luz de láser para obtener una muestra de una superficie.

Existen diferentes tipos de LIDAR, para el caso de detección de objetos con automóvil se emplea del tipo móvil terrestre.

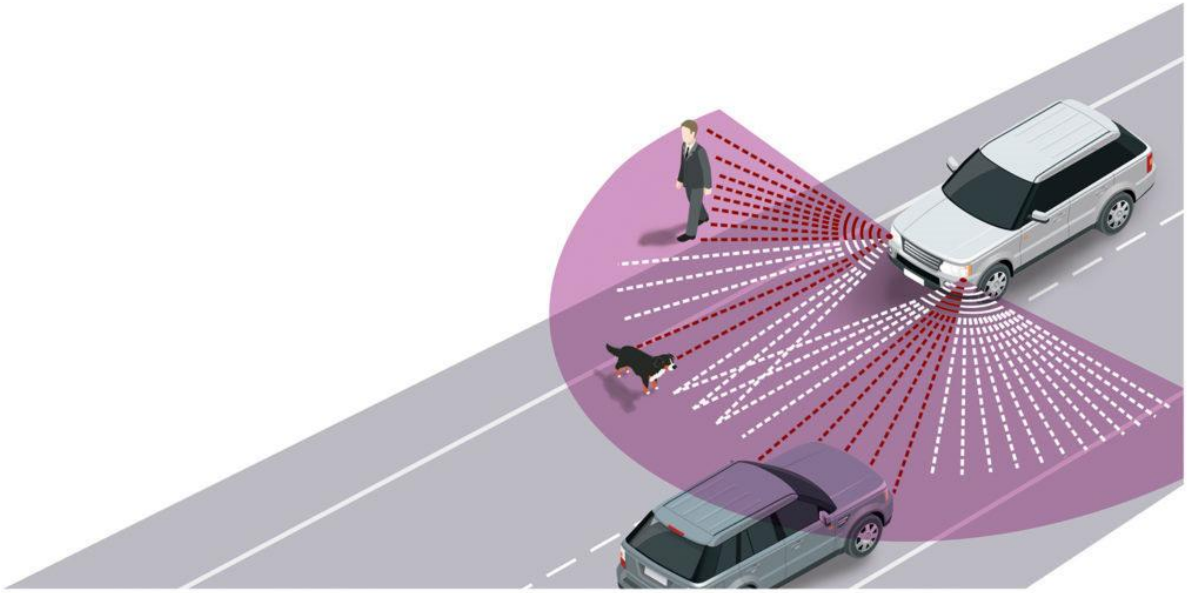


Ilustración 2-3: Ejemplo de detección de obstáculos

Está formado por un foco emisor de rayos láser infrarrojos y por una lente receptora. Mide la distancia entre el punto de emisión del láser hasta el objeto o superficie. Esta distancia es proporcional al tiempo que tarda el sensor en enviar el rayo láser y en recibirlo. Como generalmente el LIDAR se irá moviendo mientras detecta objetos, combinando la distancia estimada y la información posicional mediante un GPS, se estimará una nueva distancia que se transformará en coordenadas XYZ.

Además de la información en XYZ, se puede obtener información RGB combinando los datos del láser con los de las cámaras adicionales del sistema.

Los datos del LIDAR se procesan obteniendo una nube de puntos del entorno sobre la que se trabajará: con gran precisión se pueden reconocer los objetos y saber a qué distancia están y con qué dirección se están moviendo, de forma que se pueda prever futuras situaciones.

Para este proyecto, se emplea un LIDAR Velodyne VLP-16 [5]. Es el sensor de más baja resolución de la empresa Velodyne: 16 haces a una distancia máxima de 100 m, con una precisión en la medida de ± 3 cm. Es capaz de detectar objetos a 360º en horizontal (con una resolución de 0.1 a 0.4 º), y a ± 15º en vertical (con una resolución de 2º). Es capaz de detectar 600.000 puntos/segundo.

2.1.2 Cámaras Stereo:

Se colocan 2 cámaras que tomarán imágenes a color y en blanco y negro. Se toman imágenes cada 2 ms aproximadamente, ajustándose al tiempo de disparo del sensor LIDAR. Las cámaras de profundidad 3D producen imágenes de profundidad que sirven para crear nubes de puntos. La distancia entre los puntos no es la real, por lo que se filtran y calibran según las especificaciones del sensor. De este modo, cada pixel de la imagen en 2D se transforma en un punto en 3D.

Para realizar la transformación de 2D a 3D [6] [7], se necesitan 2 vistas del objeto. Se colocan 2 cámaras: una a la izquierda del vehículo, y otra a la derecha, de forma que se obtienen las 2 vistas. Existen diferentes algoritmos para esta transformación, pero básicamente consiste en que las imágenes son procesadas, calculando el mapa de profundidad y obteniendo así la imagen de profundidad. Para cada punto de interés del objeto, se calculan sus matrices de traslación y rotación en cada vista, y con ello, se crea el modelo 3D del objeto.

Si las cámaras son en blanco y negro, la nube de puntos resultante será sin información de color, mientras que, si las cámaras son en color, la nube de puntos resultante contendrá información del color de los objetos de la escena.

Para este proyecto, se empleará la cámara estéreo Bumblebee XB3 [8]. Cuenta con 3 sensores de 1,3 megapíxeles, con 2 líneas para el procesamiento estéreo: una de mayor resolución (ampliada) que proporciona precisión en rangos más largos, y otra de menor resolución (estrecha) que mejora la compatibilidad de rango cercano. Captura imágenes de dimensión 280x960 píxeles, con una velocidad de 16 imágenes/segundo.

Ilustración 2-4: Sensor estéreo Bumblebee XB3

2.2 ROS + RViz

ROS [9], denominado Sistema Operativo Robótico, es un software libre empleado para la programación y control de robots de manera sencilla, a partir de bibliotecas y herramientas.

Ilustración 2-5: Logo de ROS

La biblioteca está orientada para los sistemas UNIX (como Ubuntu), aunque se está extendiendo al resto de sistemas operativos.

Está formado por 2 partes:

Sistema operativo: abstracción de hardware, control de dispositivos a bajo nivel, paso de mensajes de procesos y mantenimiento de paquetes.

Basado en arquitectura de grafos: los nodos por los que está compuesto pueden mandar y recibir mensajes de distintos sensores, controles, estados, planificaciones y actuadores.

Herramientas: un conjunto de paquetes denominado *ros-pkg* que implementan funcionalidades como localización, mapeo, planificación, percepción y simulación.

2.2.1 Conceptos de ROS

A continuación, se detallan los diferentes conceptos que serán necesarios para utilizar este lenguaje.

Nodos: cada nodo tiene una única función: como pueden ser diferentes sensores que reciben información del entorno y la transforman en un tipo de datos determinados, un motor, de forma que en función de un parámetro que reciba se moverá en una dirección u otra, etc. Se compilan, ejecutan y controlan de manera individual, programados en C++ o python. Pueden publicar o suscribirse a un topic y utilizar un servicio.

Mediante la sentencia *rosrun* se puede ejecutar cada nodo de forma individual.

- Para lanzar varios nodos de forma simultanea, se usa la sentencia *roslaunch*.

Topics: sirven para enviar y recibir mensajes de diferentes tipos, de forma que los diferentes nodos se puedan comunicar entre sí.

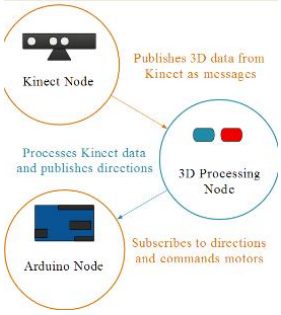


Ilustración 2-6: Comunicación entre los nodos

Cada nodo se puede suscribir a un topic mediante la sentencia ros::subscriber.

Mensajes: es un tipo de datos que sirve como comunicación entre los diferentes nodos. Generalmente, se emplean para enviar órdenes de movimiento para los diferentes robots.

Para publicar un mensaje, se emplea la sentencia ros::publisher, que sirve para que un nodo publique un mensaje a un topic.

Master: se encarga de la conexión entre nodos, permitiendo transmitir mensajes entre ellos. Es el que permite

Workspace: es un directorio donde se pueden crear y construir paquetes.

Paquetes: se encuentran dentro de un directorio de trabajo. Cada paquete está organizado en nodos que forman una interfaz ROS, y contiene todos los archivos fuentes necesarios para ejecutarlos y configurarlos.

2.2.2 Visualizador RViz

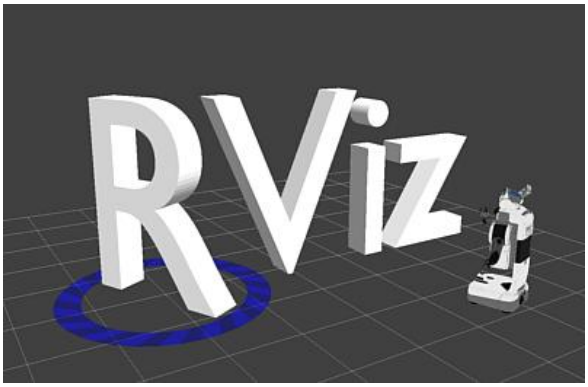


Ilustración 2-7: Logo RViz

Es una herramienta [10] de ROS que funciona como visualizador 3D empleado para mostrar modelos de robots, sus datos procedentes de los diferentes sensores, así como del entorno que le rodea en tiempo real.

Para visualizar el modelo del robot, es necesario un archivo "*.urdf*" donde se define la cinemática y dinámica del robot, la forma visual y el modelo de colisión, así como los distintos ejes y articulaciones del robot.

Añadiendo a este archivo otro de tipo "mesh", se consigue que el modelo del robot tenga elementos más realísticos. Es un archivo de tipo CAD donde se visualiza la geometría que tendrá el robot.

2.3 V-REP

V-REP [11] es un simulador robótico, empleado normalmente con uso educativo, que sirve para crear su aplicación sin depender del robot real. Coppelia Robotics ofrece este software libre como alternativa a aquellas empresas que lanzan un software exclusivo para su propio producto, permitiendo crear y diseñar diferentes modelos y simular su comportamiento.

Ilustración 2-8: Logo V-REP

Es un software muy versátil, debido a que cada modelo puede ser controlado mediante un script, plugin, nodos de ROS o APIs. Los controladores pueden estar escritos en múltiples lenguajes como C/C++, Phyton, Java, Lua o Matlab. Además, gracias al intérprete de Lua, se pueden combinar funcionalidades de alto y bajo nivel, consiguiendo una estructura sencilla de muy alto nivel.

2.3.1 Modelos en V-REP

Esta herramienta se empleará para crear los diferentes modelos de los objetos que se encontrarán durante la conducción, así como las diferentes escenas que los contengan.

Permite crear cualquier objeto a partir de formas geométricas que forman los distintos ejes y articulaciones del modelo. Mediante las diferentes herramientas, se puede seleccionar la pose de cada modelo o de cada articulación, seleccionándolo en la ventana de *"Scene Hierarchy*", así como modificar sus colores y formas*:*

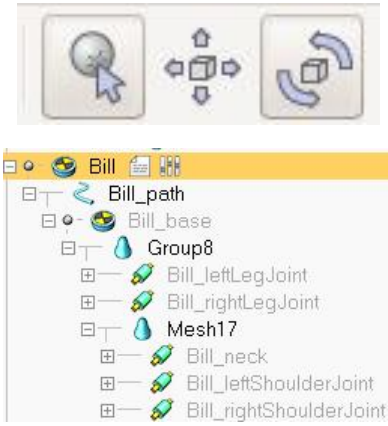


Ilustración 2-9: Componentes del modelo en V-REP

Cada modelo cuenta con varios parámetros que caracterizan su comportamiento durante simulación. Dicho comportamiento se programa mediante un script, que define la posición inicial, los distintos movimientos que realizará (independientemente de la trayectoria fijada) y el comportamiento de los diferentes sensores que compongan el modelo (en el caso del vehículo sensorizado).

Los parámetros del modelo se inicializan en la ventana "*Script Parameters*", de forma que aquellas características más importantes se puedan modificar de forma sencilla, como pueden ser velocidad, posición inicial, posición final, etc.

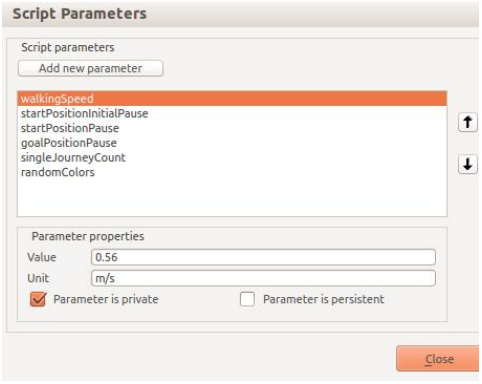


Ilustración 2-10: Parámetros del Script del modelo

Para los modelos en movimiento, se crean **Paths** o trayectorias que definen la posición que tomará el modelo durante la simulación. Para ello, en la barra de herramientas se selecciona el menú ***Add > Path > Segment type o Circle type*** en función del tipo de trayectoria que tendrá el modelo. Con el siguiente menú, se editará la trayectoria, añadiendo nuevos puntos o moviéndolos en el espacio XYZ de la misma forma que se movía el modelo:

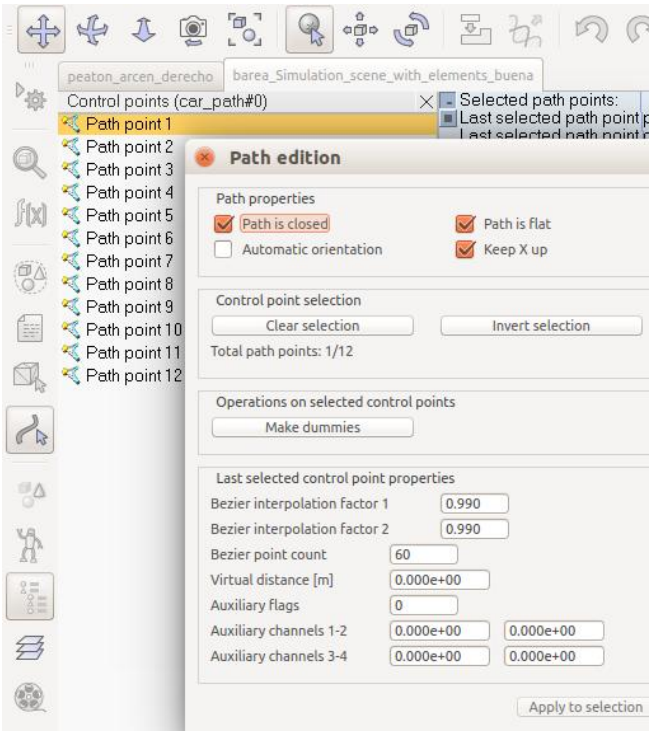


Ilustración 2-11: Edición de la trayectoria del modelo en V-REP

2.4 POINT CLOUD LIBRARY

Se trabajará con la herramienta PCL (**P**oint **C**loud **L**ibrary), que es una biblioteca para el procesamiento y tratamiento de imágenes en 2D/3D. Es un software de código abierto, gratuito para uso comercial y de investigación. Por ello, es multiplataforma: Windows, Linux, MacOS, Android e iOs.

Está dividido en un conjunto de bibliotecas más pequeñas en C++.

Este conjunto de bibliotecas trabaja con **nubes de puntos**: estructura de datos que representa un conjunto de puntos en un sistema de coordenadas tridimensional, trabajando en el sistema de coordenadas XYZ de forma que se representa la superficie externa del objeto, de modo que se tiene información de la posición y tamaño del objeto. También puede ser en 4D si se tiene información del color del objeto (en RGB o intensidad). Estas nubes de puntos son obtenidas a partir de diferentes sensores, como puede ser cámaras, escáneres 3D o láseres.

2.4.1 Clase PointCloud

Para las nubes de puntos, existe una clase denominada ***pcl::PointCloud < T >*** de tipo vector. T define de qué tipo será la nube de puntos:

PointXYZ: representa solamente información XYZ, es decir, las coordenadas 3D de cada punto. También existe el tipo XY para información en 2D.

PointXYZI: añade información sobre la intensidad de cada punto.

PointXYZRGBA: añade información de color RGBA, de tipo entero.

PointXYZRGB: añade información de color RGB, pero de tipo float.

En este proyecto, se trabajará con el tipo XYZ y XYZRGB.

La nube de puntos, además del tipo de datos que contenga, viene definida por los siguientes elementos:

Width: especifica el ancho de la nube de puntos para nubes organizadas, o el número total de puntos en la nube para nubes desorganizadas.

Heigth: especifica la altura de la nube de puntos para nubes organizadas, o se fija con valor 1 para nubes de datos desorganizadas.

Points: indica el tipo de datos en el que se almacena la nube de puntos (descritos anteriormente como T).

Is_dense: indica si todos los datos de los puntos son finitos (true) o contiene valores Nan (false).

sensor_origin: especifica la pose del sensor desde el que se capturan los datos. Este valor es opcional.

Sensor_orientation_: especifica la rotación del sensor. Este valor es opcional.

2.4.2 Módulos de PCL

A continuación, se explicarán algunos de los principales módulos de PCL que se utilizarán para este proyecto:

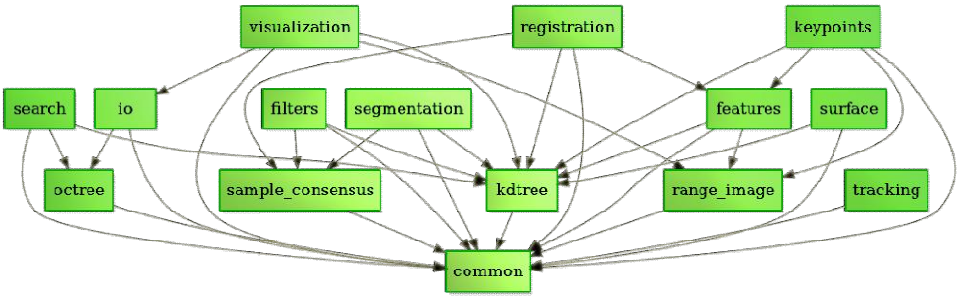


Ilustración 2-12: Conexión de los módulos de la herramienta PCL

libpclsegmentation: contiene algoritmos que sirven para segmentar una nube de puntos: la nube de puntos se separa en subgrupos para luego procesarlos de forma independiente.

Libpclfilters: contiene algoritmos que permiten filtrar la nube de puntos, ya sea eliminando valores fuera de un rango de longitud determinado (para analizar una zona específica), eliminar las proyecciones de los objetos o datos atípicos.

Los filtros usados en este proyecto son: ***VoxelGrid filter*** y ***Extracting indices***.

LibpclKdtree: se almacena en una estructura de tipo árbol los puntos en k-dimensiones con el fin de realizar una búsqueda para encontrar un objeto similar cercano al objeto de análisis.

Libpclio: las nubes de puntos se almacenan en archivos PCD (Point Cloud Data). Esta biblioteca permite leer y almacenar este tipo de datos, de forma que se pueda trabajar con ellos. También sirve para extraer una nube de puntos de un sensor.

Libpclvisualization: con este módulo se puede visualizar la nube de puntos en 3D, pudiendo visualizar también el color, ajustar el ángulo de la cámara para visualizar la nube total o una parte de interés.

Libpclfeatures: este módulo contiene diferentes tipos de estructuras que permiten asignar una función geométrica para una nube de puntos 3D. Un ejemplo de este módulo es el de los descriptores para puntos en 3D, de forma que son capaces de describir, dados unos puntos en el espacio 3D, la forma geométrica de estos para así poder identificar de qué objeto se trata.

2.4.3 Archivos .pcd

Como se ha nombrado anteriormente, las nubes de puntos resultantes de los diferentes sensores, se transformarán a formato .pcd (*P*oint *C*loud *D*ata) para trabajar con la herramienta PCL. Estos archivos contienen toda la información de la nube de puntos. A continuación se muestra un ejemplo de una nube de puntos almacenada en un archivo .pcd:

```
1 # .PCD v0.7 - Point Cloud Data file format
2 VERSION 0.7
3 FIELDS x y z rgb
4 SIZE 4 4 4 4
5 TYPE F F F F
6 COUNT 1 1 1 1
7 WIDTH 149
8 HEIGHT 1
9 VIEWPOINT 0 0 0 1 0 0 0
10 POINTS 149
11 DATA ascii
12 9.9865837 -0.62213206 -1.1134225 0
13 9.736989 -0.60880488 -1.0167459 4.4251618e-39
14 10.098609 -0.62789053 -1.0901927 1.1490647e-43
15 9.6220503 -0.6026721 -0.97076327 8.8497071e-39
16 9.4215021 -0.59205908 -0.91728276 8.5734789e-39
17 9.6004896 -0.26432219 -0.90006644 2.3261555e-43
18 9.336956 -0.58749688 -0.85970229 1.4754033e-39
19 9.352005 -0.26006663 -0.81087297 3.0688436e-43
20 9.5337687 -0.2631906 -0.86021054 3.0828566e-43
21 8.9048872 -0.56460708 -0.71016067 2.760558e-43
22 9.0370283 -0.57159269 -0.75248998 2.760558e-43
23 9.1734352 -0.57880956 -0.79613817 2.760558e-43
24 9.0791569 -0.25539398 -0.72331846 3.0688436e-43
25 9.2131653 -0.25768784 -0.76640517 1.1066446e-39
26 8.6645651 -0.856673 -0.63078922 2.6624671e-43
27 8.7314281 -0.86265582 -0.66634089 2.6624671e-43
28 8.7451687 -0.55616003 -0.65136528 2.760558e-43
29 8.7523594 -0.24982266 -0.60501337 3.0268047e-43
30 8.8805799 -0.25201318 -0.64506757 3.0268047e-43
```

Ilustración 2-13: Contenido archivo pcd de nube de color azul.

Se procede a analizar los parámetros del archivo:

Versión: en la primera línea, aparece la versión del archivo .pcd, por lo que para trabajar con él la librería deberá tener al menos dicha versión.

Fields: describe el tipo de nube de puntos. La nube de puntos es tridimensional (XYZ), pero también puede tener información de color o de intensidad. En este campo, se especifica el tipo de nube a procesar.

Size: tamaño del campo en bytes. Para el tipo float es 4.

Type: indica el tipo de datos de cada variable. Puede ser de tipo ***I***(signed), ***U*** (unsigned) y ***F*** (float). En este caso es F, por lo que los datos están almacenados en tipo Float.

Count: indica el número de elementos que tendrá cada dimensión, en este caso, 1.

Width: indica el ancho de la nube de puntos. En caso de tener una nube desordenada, indicará el número total de puntos (como es en este caso).

Heigth: indica la altura (número total de filas) de la nube de puntos organizada. Se observa que el valor es 1, debido a que la forma de la nube es desordenada.

Viewpoint: indica el punto de vista con el que es observada la nube de puntos, formada por la traslación (en X, Y, Z) y un cuaternio (qw qx qy qz).

Points: indica el número total de puntos, siendo igual a *width·height*.

Data: indica el valor en formato ***ascii*** de cada punto, con todas sus coordenadas.

A continuación, se toma la información de una nube de puntos de color rojo, para observar la diferencia en el campo rgb con respecto a la anterior de color azul:

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 29
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 29
DATA ascii
10.970217 -0.70253456 -3.2242441 1.0561082e-38
10.944601 -0.70064467 -3.1752024 2.03882e-38
11.478646 -0.31015018 -2.5592847 9.8263981e-39
11.261399 -0.30630699 -2.4489968 1.7356909e-38
11.216 -0.30541953 -2.3570991 1.9836467e-38
11.186213 -0.30477071 -2.2286389 1.9836467e-38
11.205358 -0.30516538 -2.2935767 1.9836467e-38
11.173092 -0.30446732 -2.1449294 1.9836467e-38
10.823591 -0.68338001 -2.0395579 1.8734801e-38
11.169609 -0.30432588 -2.0432279 1.9836467e-38
10.817401 -0.68236387 -1.9406886 1.9744992e-38
11.188896 -0.30460364 -1.966022 1.9836467e-38
11.209535 -0.30493626 -1.9292848 1.9836467e-38
11.247378 -0.30555031 -1.8549032 1.9836467e-38
11.291011 -0.30627209 -1.7811595 1.9836467e-38
11.32007 -0.30675751 -1.7249528 1.9836467e-38
11.354601 -0.30733994 -1.6490877 1.9836467e-38
11.427651 -0.30859661 -1.5781918 1.855077e-38
11.521791 -0.31022415 -1.5297164 1.2214121e-38
11.647118 -0.31239173 -1.50495 1.0102265e-38
10.800055 -0.67859524 -1.4348046 1.5520199e-38
```

Ilustración 2-14: Contenido del archivo pcd de una nube de puntos roja.

Para este caso, el valor del campo rgb es menor.

Los descriptores también se pueden almacenar en este tipo de archivos. En el campo *Field* especifica el tipo de descriptor que es, y en *Count* el número de contenedores del descriptor. Al no ser una nube de puntos, el tamaño especificado en *Points* es de 1, ya que en una misma gráfica se especifica el valor de cada contenedor. A continuación, se muestra un ejemplo de un descriptor VFH:

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS vfh
SIZE 4
TYPE F
COUNT 308
WIDTH 1
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 1
DATA ascii
100.62876 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 100.62876
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 100.62876 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Ilustración 2-15: Contenido del archivo pcd de un descriptor VFH.

3 DESCRIPTORES VFH

Una forma de reconocer objetos 3D es mediante la búsqueda de correspondencias entre dos nubes de puntos.

Una nube de puntos puede contener información de cada coordenada en XYZ, color RGB o intensidad de cada punto, pero esta información no es suficiente: al comparar estas características entre 2 nubes de puntos se observa que, aunque haya puntos de cada nube que compartan las mismas coordenadas o características, en conjunto no representan el mismo objeto.

Una forma más precisa de identificar objetos es mediante el uso de descriptores, que contienen mayor información de las características geométricas de la nube de puntos. Estas características se representan en forma de histogramas, que son una forma de representación gráfica de la distribución de una variable con respecto a una característica.

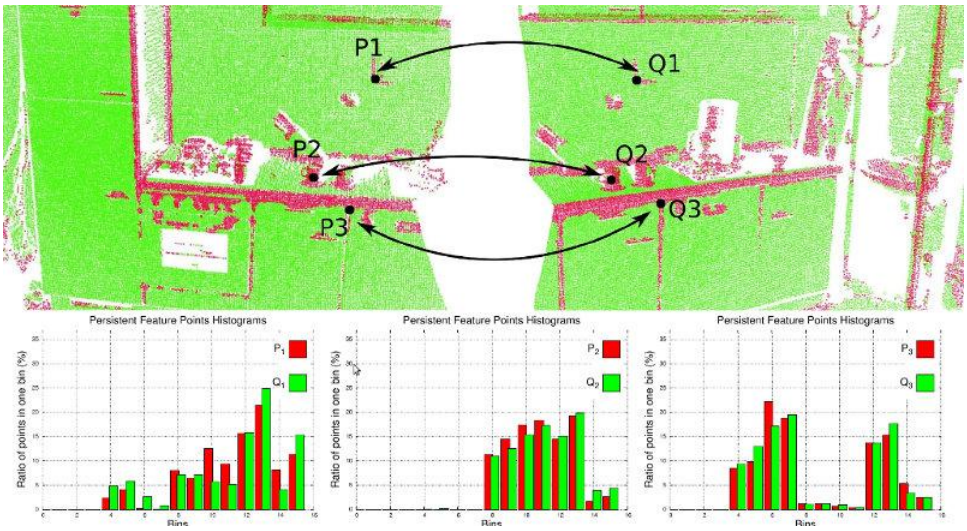


Ilustración 3-1: Comparativa entre 2 nubes de puntos.

Para el reconocimiento de objetos de este proyecto, se utilizará el descriptor VFH. Es una extensión del histograma FPFH.

En los siguientes apartados, se explicará el funcionamiento de los descriptores en los que está basado el descriptor VFH.

PFH: Point Feature Histogram

Es uno de los descriptores base. Para este caso, la información sobre la geometría de la nube alrededor de un punto por el análisis entre las direcciones de las normales de los puntos vecinos dentro de un radio r.

Se estiman las normales de cada punto, y se captura la variación de la superficie de la muestra teniendo en cuenta las interacciones entre dichas normales.

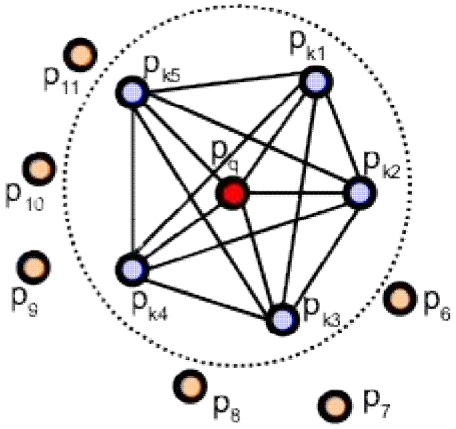
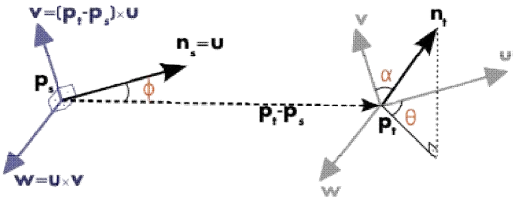


Ilustración 3-2: Interacciones de p_q con sus pares de vecinos en un radio r.

Para ello, se toman dos puntos *p* y *q*, se crean 3 vectores unitarios (u, v, w) centrados en *p* que forman un espacio vectorial. La diferencia entre las normales de *p* y *q* se representa mediante 3 ángulos siguiendo las siguientes expresiones:



$$u = n_i \qquad v = u \times \frac{(p_j - p_i)}{||p_j - p_i||_2} \qquad w = u \times v$$

$$\alpha = v \cdot n_j \qquad \phi = u \cdot \frac{(p_j - p_i)}{d} \qquad \theta = \arctan\left(w \cdot n_j, u \cdot n_j\right)$$

Ilustración 3-3: Ecuaciones de las componentes del descriptor.

Los 3 ángulos y la distancia d ($d = || p - q ||_2$) se computan para cada par de *k*vecinos del punto *p*.

Se obtiene una complejidad de un espacio vectorial de nk^2 (siendo n el número de puntos a analizar y k los vecinos de dichos puntos), lo que hace que el análisis sea muy lento.

FPFH: Fast Point Feature Histogram

Debido a la complejidad del descriptor PFH, *Rusu* propone como alternativa un sistema más rápido Este descriptor está basado en PFH, pero solo considera las conexiones entre los puntos clave y sus vecinos, evitando así las interconexiones. Se reduce el espacio vectorial a nk dimensiones, y se consigue aumentar la velocidad de procesado.

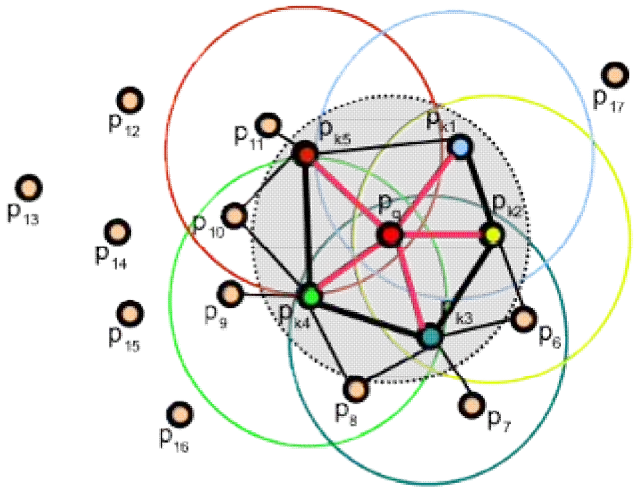


Ilustración 3-4: Interacciones de p_q con sus vecinos en un radio r.

Se construye un PFH simplificado (SPFH): la diferencia de este caso es que se computan los 3 ángulos y sus *k*vecinos más cercanos en lugar de los pares de *k*vecinos, y se almacenan en 3 histogramas separados. El descriptor para un punto p_q se calcula como la de su SPFH y el sumatorio de cada SPFH con sus *k*vecinos:

$$FPFH\left(p_q\right) = SPFH\left(p_q\right) + \frac{1}{k}\sum_{i=1}^{k} w_k \cdot SPFH\left(p_k\right)$$

Ilustración 3-5: Ecuación del descriptor para un punto p_q.

En la ecuación anterior, ω_k se corresponde con la distancia existente entre el punto analizado y cada vecino.

VFH: Viewpoint Feature Histogram

Para el descriptor VFH [1] se añade a FPFH el punto de vista del vector director como una nueva componente. Este descriptor está compuesto por una componente de la dirección del punto de vista y otra de la forma de su superficie.

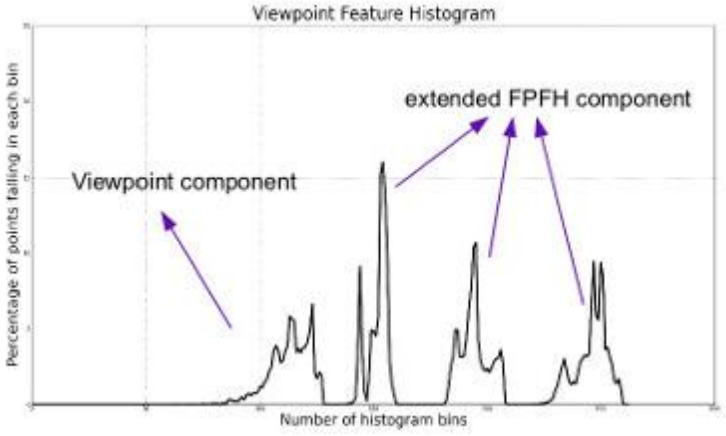


Ilustración 3-6: Histograma VFH.

Para ello se calcula el centroide del objeto. Se normaliza el vector entre el sensor y el centroide. Se calcula el ángulo entre el vector normalizado y la normal de cada punto, agrupándose el resultado en el histograma.

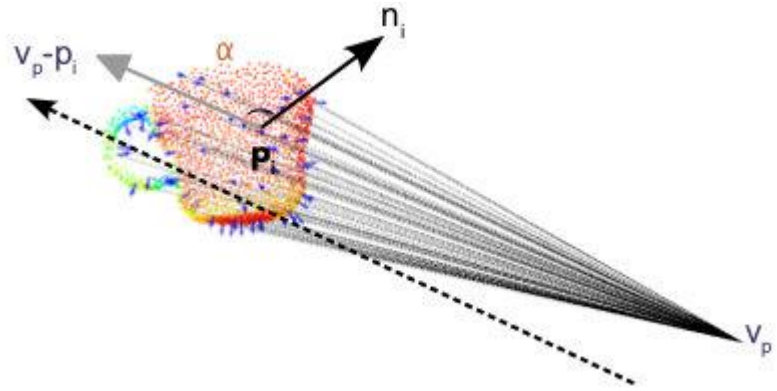


Ilustración 3-7: Componentes para un punto p_i respecto un punto de vista v_p.

Para la componente de la forma de la superficie, se calcula el FPFH respecto del centroide del objeto con cada una de las normales de la superficie del objeto.

Cada componente en 3D se almacena en un sub-histograma de 45. El ángulo del punto de vista central se traslada a cada normal codificándolo en un histograma de 128, y las distancias entre cada punto y el centroide se almacena en un histograma de 45. En total tiene una capacidad de 308.

Este histograma es capaz de realizar un reconocimiento de alta calidad y de baja complejidad *O(n),* aunque es bastante sensible al ruido.

A continuación, se muestra una tabla con la comparación entre los diferentes descriptores [12] nombrados anteriormente:

Descriptor	Tipo	Tamaño	Info. color
PFH	Local	125	No
FPFH	Local	33	No
VFH	Global	308	No

Tabla 2: Comparativa entre los diferentes descriptores estudiados.

La herramienta PCL dispone de funciones para trabajar con estos descriptores y emplearlos para reconocer objetos a partir de nubes de puntos. Dado un punto de la nube, existen 2 formas de buscar los vecinos con esta herramienta, empleando consultas kd-tree:

Buscando todos los vecinos dentro de un radio *r* dado por el usuario.

Buscando los *k* vecinos fijando este número.

4 CLASIFICACIÓN DE OBJETOS MEDIANTE VFH

El objetivo de este apartado es realizar el reconocimiento de los objetos que pueden estar presentes durante la circulación de un vehículo empleando los descriptores VFH.

Se realizarán 2 bases de datos con los 2 métodos de adquisición de datos que posee el vehículo autónomo. El primero es a partir del LIDAR, con datos “spare”, de menor calidad debido a que solo posee 16 haces y se pierde información. El segundo, a partir de las cámaras se convertirá la imagen de profundidad en una nube de puntos, simulando un LIDAR de 64 haces, con datos “good”, pues se pierde menos información.

Para ello, se emplearán una serie de algoritmos donde se procesarán las diversas nubes de puntos, se separarán en objetos individuales y se procesarán de forma que se forme la base de datos.

Base de datos VLP16: de cada objeto (coche, árbol, peatón), se tomará una muestra con un giro sobre el eje Z cada 5º, de forma que se obtienen 144 imágenes.

Base de datos HDL64E: al tener mejor resolución, se tomará muestras cada 10º, obteniendo 72 imágenes de cada tipo de obstáculo.

4.1 BASE DE DATOS PEATÓN

Para cada uno de los sensores, se guardará la nube de puntos en formato .pcd del peatón. Para ello, se girará el peatón sobre su eje Z, de forma que se capture en diferentes posiciones.

La distancia entre el vehículo y el peatón a capturar es de 3 m (en el eje X). Se elige esta distancia para capturar el mayor número de puntos de la nube en el caso "spare" (ya que, al lanzar solo 16 haces, hay pocos que alcanzan al objeto):

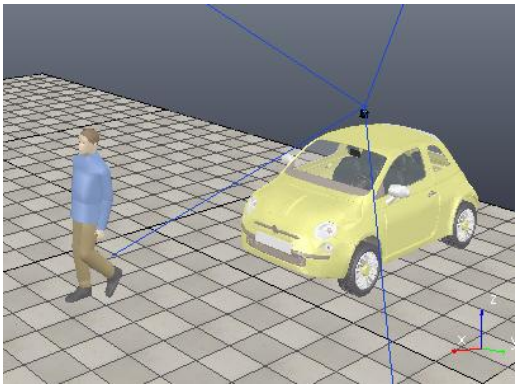


Ilustración 4-1: Modelo peatón a 0º en VREP.

A continuación, se muestran varios ejemplos del giro del objeto en la base de datos en el entorno de simulación VREP y de cómo el sensor VLP16 lo detecta en el visualizador de ROS RViz, cada 90º, capturándolo a 0º, 90º, 180º y 270º:

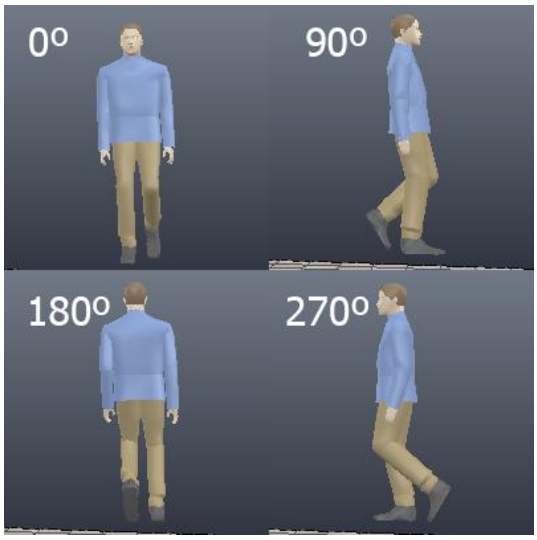


Ilustración 4-2Rotación del peatón en V-REP

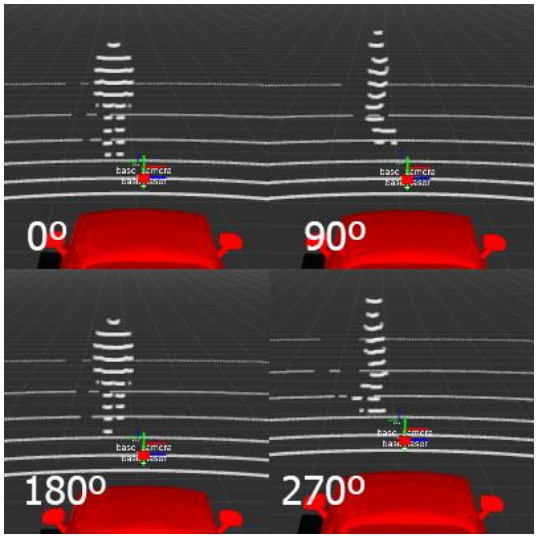


Ilustración 4-3: Rotación del peatón en RViz

4.2 BASE DE DATOS COCHE

Para cada uno de los sensores, se guarda la nube de puntos en formato .pcd de un vehículo. Sobre el eje Z, se rota el vehículo sus 360º capturando así diferentes posibles posiciones.

La distancia entre el vehículo sensorizado y el vehículo capturado es de 9 m (en el eje X). Se elige esta distancia para capturar el mayor número de puntos en el caso "spare":

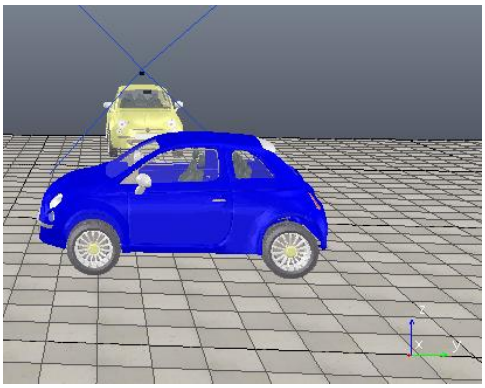


Ilustración 4-4: modelo vehículo a 0º en VREP.

A continuación, se muestran varios ejemplos de la base de datos del coche, tanto en el entorno de simulación VREP como en el visualizador de datos 3D RViz de los datos capturados por el lidar VLP16. Se hace una rotación cada 90º, de modo que se obtiene la visualización en 0º, 90º, 180º y 270º:

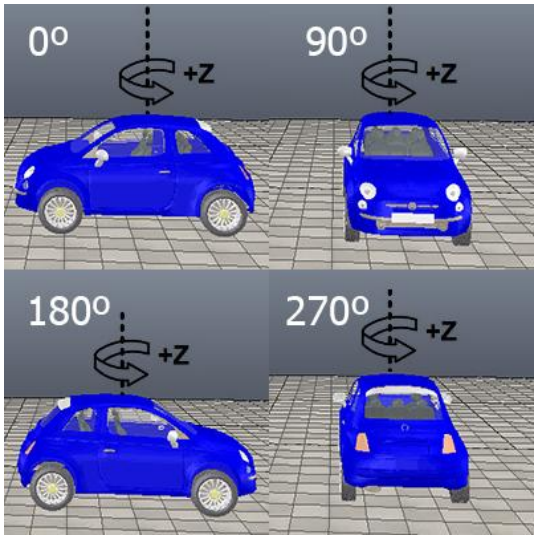


Ilustración 4-5: Rotación del vehículo en V-REP

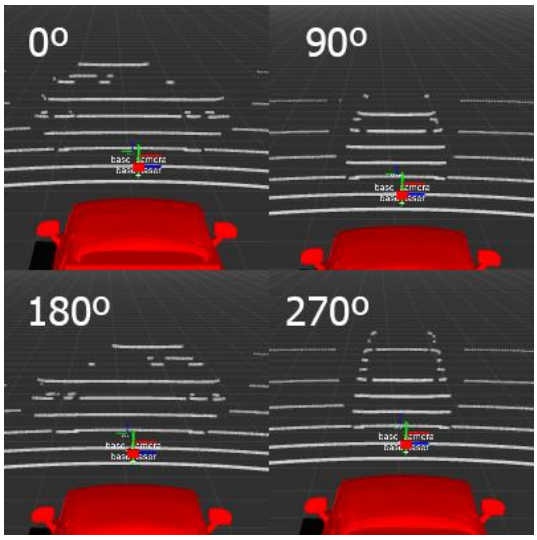


Ilustración 4-6: Rotación del vehículo en RViz

4.3 OBTENCIÓN DE NUBES DE PUNTOS:

En este apartado, se explicarán los principales algoritmos que será necesarios para obtener la nube de puntos a partir de los sensores del vehículo.

4.3.1 Smart Elderly Car en Rviz:

El primer paso es lanzar el archivo que permitirá visualizar y controlar el vehículo sensorizado. Para ello, se emplea el archivo "*joy_navegation.launch*" del paquete smart_eld_car.

Este archivo contiene otros nodos de ROS en C++ que configuran los parámetros del vehículo: el primero de ellos *(local_transform_velodyne*), se encarga de realizar las transformaciones necesarias para los sistemas de referencia, y el segundo (*GUI_robot_joy*) se encarga de configurar un Joystick para poder controlar el vehículo de forma opcional.

Se abre el modelo en Rviz: para ello es necesario el archivo *velodyne_cam.rviz* y el archivo *smartElderlyCar.urdf*. De esta forma se carga el modelo visual y geométrico del vehículo y se configuran los sensores en programa Rviz.

4.3.2 Conversión imagen de profundidad – Point Cloud

Existen dos formas de obtener la nube de puntos del entorno que se procesará: la primera es a través del láser escáner Velodyne, que mediante los haces lanzados es capaz de formar la imagen 3D del entorno; y la segunda es mediante las cámaras de video.

Se coloca una cámara a la izquierda del vehículo y otra a la derecha, de forma que se obtenga una imagen a color y otra en blanco y negro de cada lado.

Mediante el archivo "*convert_depth_to_pointcloud.launch*", se procesa la información de la cámara y se obtiene una nube de puntos con la que se trabajará.

Este archivo recibirá como entradas:

- ***Sensor_msgs/CameraInfo***: mediante la suscripción a este topic de la cámara, se obtienen los parámetros de calibración. Estos parámetros sirven para, mediante la obtención de una imagen, calcular las diferentes distancias entre los objetos presentes en ella.
- ***Sensor_msgs/Image:*** mediante la suscripción a este topic de la cámara, se obtiene la imagen de tipo RAW. Este tipo de archivo contiene la totalidad de los datos de la imagen, ofreciendo así una mayor profundidad de color y, por tanto, una mayor calidad de los resultados. Mediante el paquete *depth_image_proc,* que contiene diferentes nodos para procesar imágenes de profundidad, se realizará la obtención de la nube de puntos.

  Para ello, se emplea el nodo *"point_cloud_xyz.cpp"*. Este nodo obtiene la nube de puntos en 3D:

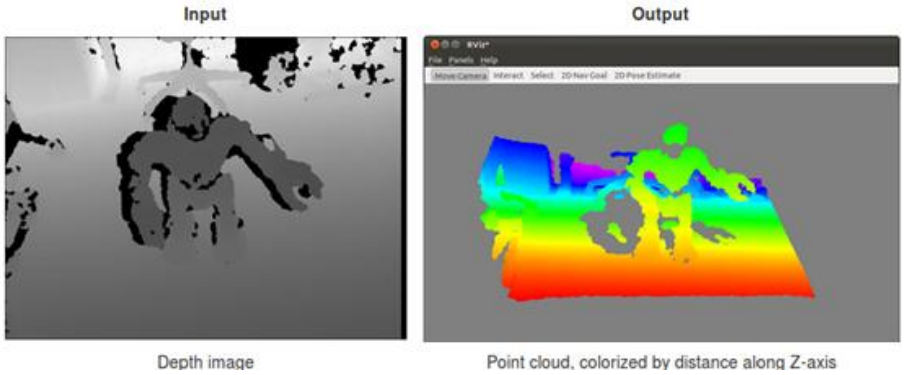


Ilustración 4-7: Obtención de una imagen de profundidad

La nube de puntos resultante será la salida del algoritmo, de forma que se podrá acceder a ella subscribiéndose al topic "***points***".

4.3.3 Procesamiento de nubes de puntos RGB:

En algunas ocasiones, es interesante trabajar con nubes de puntos con información de color para poder segmentar y separar los objetos.

Para ello, se combina la información de una nube de puntos (ya sea obtenida directamente por un sensor láser escáner o tras procesar una imagen de profundidad), con la información de las cámaras de video con color.

Se utiliza el archivo "*coloring.launch*", del paquete but_calibration_camera_velodyne, que carga dos archivos de tipo ".yaml":

- *Calibration:* utiliza la información de calibración de la cámara (*camera_info*) junto con la imagen a color en formato RAW. Se obtiene una nube de puntos, almacenada en el topic */velodyne_points_local*.
- *Coloring:* se encarga de orientar la nube de puntos obtenida a color en 6DoF: con una rotación en los 3 ejes, combinado con la rotación sobre los 3 ejes perpendiculares. La nube reorientada se almacena en el topic */velodyne_colored_points*.

4.4 CREACIÓN DE LAS BASES DE DATOS:

El propósito de este apartado es descomponer una nube de puntos de una escena en diferentes **clusters** (subgrupos de puntos), de forma que se puedan analizar por separado, almacenar y clasificar.

Cluster_extraction.cpp

Este algoritmo segmentará los distintos objetos de una escena a partir del método Euclidean Cluster Extraction. Para ello se empleará la clase de la biblioteca PCL `pcl::EuclideanClusterExtraction`, basado en los siguientes puntos:

El tiempo de procesado de una nube de puntos *P* no organizada con **n** objetos es muy elevado debido a la cantidad de información presente en ella. Este tiempo se puede reducir significativamente dividiendo *P* en partes más pequeñas, es decir, segmentando la nube de puntos.

Empleando el método euclidiano, se realiza una subdivisión del espacio usando una estructura octree (empleando restricciones de volumen y separación de los vecinos más cercanos).

En primer lugar, para obtener la nube de puntos de una escena procedente del sensor (ya sea el lidar o las cámaras), en la función main() hay que suscribirse al topic donde se almacenan los datos procedentes de él. El topic se llama ***/velodyne_points_local*** y los datos se envían a la función ***cloud_cb***, que recibe una constante de tipo sensor_msgs::PointCloud2ConstPtr, donde se encuentran los datos del sensor.

La nube inicial se filtra mediante un Voxel Filter, a través de la siguiente sentencia:

```
vg.setLeafSize (0.1f, 0.1f, 0.1f);
```

Los números introducidos implican que se filtra con un tamaño de 10 cm (introducidos en metros como coma flotante).

Los pasos a seguir, una vez obtenida una nube de puntos con numerosos objetos son los siguientes:

1. Crear una representación Kd-tree para dicha nube de puntos.
2. Se establece una lista vacía de clusters C, y unos puntos Q que deben ser comprobados.
3. Para cada punto de la nube pi perteneciente a P:

- Se busca el conjunto Pik de puntos vecinos de pi en un radio fijado.
- Se comprueba si cada Pik ha sido procesado anteriormente, y si no es así, se añade al conjunto Q.

  Una vez procesados todos los puntos del conjunto Q, se añaden a un cluster C y se resetea el conjunto Q.

4. El algoritmo finaliza una vez que todos los puntos de la nube han sido agrupados en diferentes clusters C.

El algoritmo permite fijar un volumen mínimo y máximo para los clusters con las siguientes sentencias, donde se introducirá el mínimo y máximo número de puntos:

```
ec.setMinClusterSize (min);
ec.setMaxClusterSize (max);
```

Y un radio de búsqueda de los vecinos más cercanos, es decir, la distancia máxima que se considerará que un punto pertenece a un cluster o no en cm:

```
ec.setClusterTolerance (tolerance);
```

Se jugará con dichos valores:

Máximo y mínimo número de puntos: con estas restricciones, los objetos muy pequeños, como pueden ser sombras o tramos de carretera, se pueden filtrar.

Tolerancia: Si el valor de la tolerancia es muy pequeño, puede ocurrir que un solo objeto sea considerado como múltiples objetos, y por el contrario, si es un valor muy grande, varios objetos se pueden considerar como uno solo.

A este algoritmo, se le añade una función que sirve para almacenar cada cluster en archivos .pcd, de forma que se pueda trabajar con ellos en forma de base de datos:

```
writer.write<pcl::PointXYZ> (ss.str (), *cloud_cluster, false);
```

Donde el primer elemento de la función será el nombre con el que se guardará el archivo, introduciéndolo de la forma, para el elemento número j:

```
ss << "cloud_ejemplo" << j << ".pcd";
```

El segundo elemento, es un cluster de la nube procesada.

A continuación, se muestra un ejemplo de una escena con varios coches en VREP, y como son separados en diferentes clusters, de forma que se obtiene cada objeto de forma individual. En el visualizador, cada cluster se separa dentro de un cubo de color de forma que se identifica cada objeto por separado:

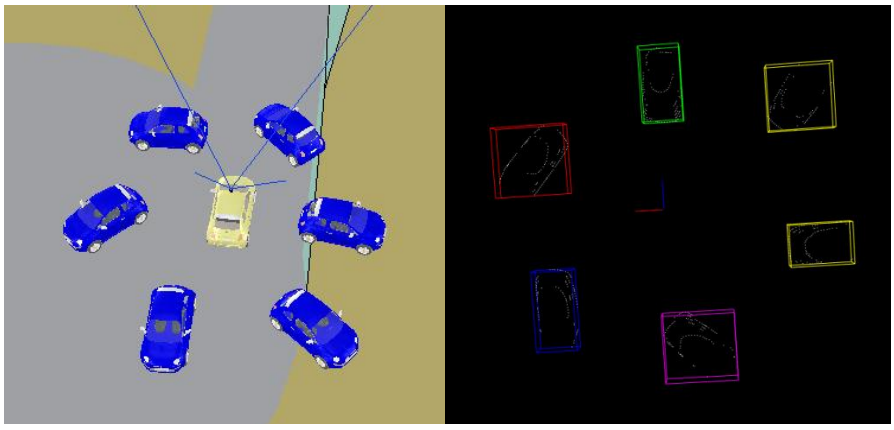


Ilustración 4-8: Separación en clusters de una escena

pcd_read.cpp:

Este algoritmo abre un archivo .pcd almacenado y lo visualiza, de forma que si es válido se almacena con el nombre deseado, y si no se descarta empleando la terminal.

Servirá para crear la base de datos a partir de los clusters almacenados, guardando así aquellos clusters con datos válidos.

concatenate_pcl.cpp

En ocasiones, debido a la separación con la que se procesa la escena (debido al número de haces del sensor, pues cuanto más pequeño menos información del objeto se obtiene), un mismo objeto se considera como 2 clusters:

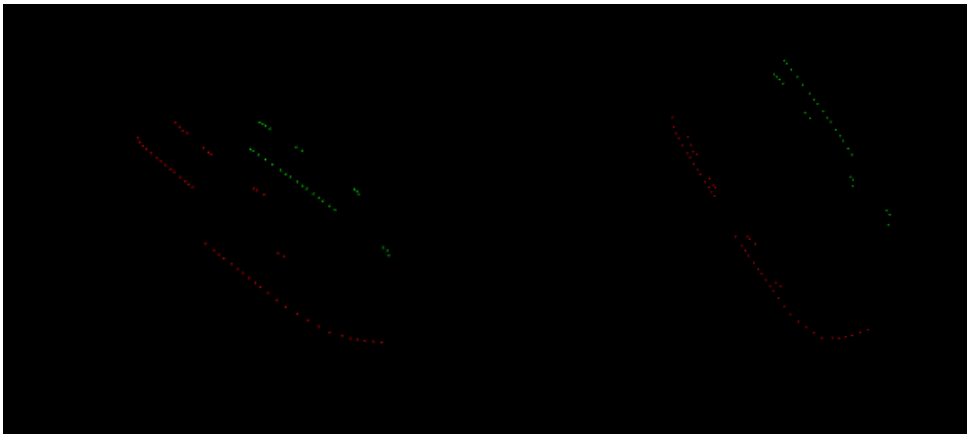


Ilustración 4-9: 2 clusters pertenecientes a un mismo vehículo.

Para no modificar el valor de tolerancia del método de extracción, se genera un algoritmo que reconstruya el objeto.

Este algoritmo recibe 2 nubes de puntos a y b, que inicialmente pertenecen a un mismo objeto, pero que han sido segmentadas y almacenadas como 2 nubes. Con las siguientes sentencias, se suman o concatenan a y b, obteniendo una nueva nube resultado c que contendrá las 2 nubes:

```
*cloud_c=*cloud_a;

*cloud_c+=*cloud_b;
```

scale_pointcloud.cpp

Una vez generada la base de datos de cada objeto, se escalarán todos para que tengan una componente en común, de modo que la comparación geométrica sea más exacta.

Se trabajará en la longitud Z (altura de la nube de puntos), y se hará que todos los objetos de la base de datos tengan la misma altura. De este modo, si en la escena se encuentran vehículos más altos, o peatones más bajos, se transformarán para que todos sean iguales.

Se calcula el valor de N, que funcionará como un divisor o multiplicador de la nube de puntos de entrada. *long_z* será la longitud final deseada común a todos los clusters, y z_max y z_min la longitud inicial del cluster, en este caso tomará un valor de 5.0:

```
N=5.0/(z_max-z_min);
```

A partir de una matriz de transformación, se modifica el cluster con el valor calculado, de forma que ahora el cluster tendrá la longitud en Z deseada (siendo proporcional en X e Y, de forma que el objeto no quede deformado).

```
Eigen::Matrix4f transform = Eigen::Matrix4f::Identity();

transform (0,0) = transform (0,0) * N;

transform (1,1) = transform (1,1) * N;

transform (2,2) = transform (2,2) * N;

pcl::transformPointCloud (*cloud_cluster_xyz, *cloud_cluster_xyz, transform);
```

4.5 OBTENCIÓN DE LOS DESCRIPTORES VFH

VFH (Viewpoint Feature Histogram) es uno de los métodos existentes para el reconocimiento de objetos y de su posición, a partir del método de descriptores FPFH. A partir de un grupo de puntos denominado *cluster,* se estima y calcula de forma estadística la dirección del origen y las normales estimadas en cada punto del cluster. Se genera un histograma formado por el ángulo entre el origen central trasladado a cada normal.

4.5.1 Reconocimiento de cluster y estimación de pose 6DOF a partir de los descriptores VFH

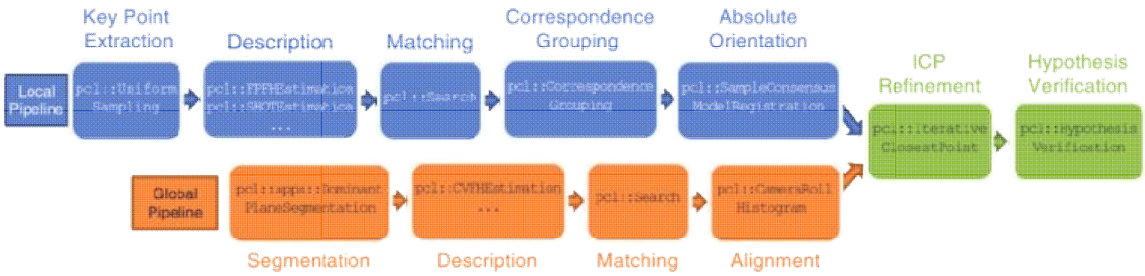


Ilustración 4-10

Con este método se obtendrán los posibles candidatos similares al objeto buscado. Se realizará una búsqueda de los objetos cercanos que sean más parecidos a partir de las similitudes geométricas entre las nubes de puntos que formen.

Dentro de este procedimiento se encuentran 2 etapas:

4.5.2 Entrenamiento:

Con los diferentes objetos ya separados como un grupo de puntos, se obtiene la posición, y girando alrededor del objeto con respecto a la posición de la cámara (origen), se calcula un

descriptor VFH para cada una de las vistas. Se almacenan las vistas y se construye una representación kd-tree.

4.5.3 Test:

A partir de distintas escenas, una vez entrenado con distintas cluster, se calcula el descriptor VFH desde la posición a la cámara, y con ellos, se buscan los candidatos en el kd-tree entrenado.

Se utilizarán 3 algoritmos, uno para generar el descriptor VFH y otros 2 para realizar distintas pruebas.

PCL_VFH_loop_extraction.cpp

En primer lugar, se generan el histograma de cada uno de los archivos pcd que forman parte de la base de datos.

La herramienta PCL incluye módulos para poder trabajar con el descriptor VFH. Para ello, son necesarias las bibliotecas ***"pcl/features/normal_3d.h"***, ***"pcl/features/vfh.h"*** y para visualizar el histograma ***"pcl/visualization/histogram_visualizer.h"***.

En este código, mediante un bucle, se leen todos los *clusters* almacenados en la base de datos de los cuales se quiere obtener su histograma:

```
file_pcd << "peaton_" << i << ".pcd";
```

Con esta sentencia se lee el archivo "*peaton_i_j.pcd*", siendo i el ángulo de giro del objeto peatón.

```
if (pcl::io::loadPCDFile<pcl::PointXYZ> (file_pcd.str (), *object) ==
                        -1) //* load the file
```

Con esta sentencia, se comprueba si existe el archivo nombrado anteriormente, y si existe, se almacena en *object*, del tipo *pcl::PointCloud<pcl::PointXYZ>::Ptr*.

A continuación, empleando las diferentes sentencias del módulo de descriptores, se estiman las normales y se obtiene el descriptor VFH, el cual se muestra en pantalla.

Para estimar las normales, se emplea el método de los vecinos dentro de un radio fijado, el cual se puede modificar con la siguiente sentencia:

```
normalEstimation.setRadiusSearch(radius);
```

Se prueba variando el radio de búsqueda: al estar las nubes escaladas con una altura de 5 metros, cuanto mayor sea el radio de búsqueda, más información del objeto se obtiene. Esta diferencia sólo es apreciable en la base de datos de los coches:

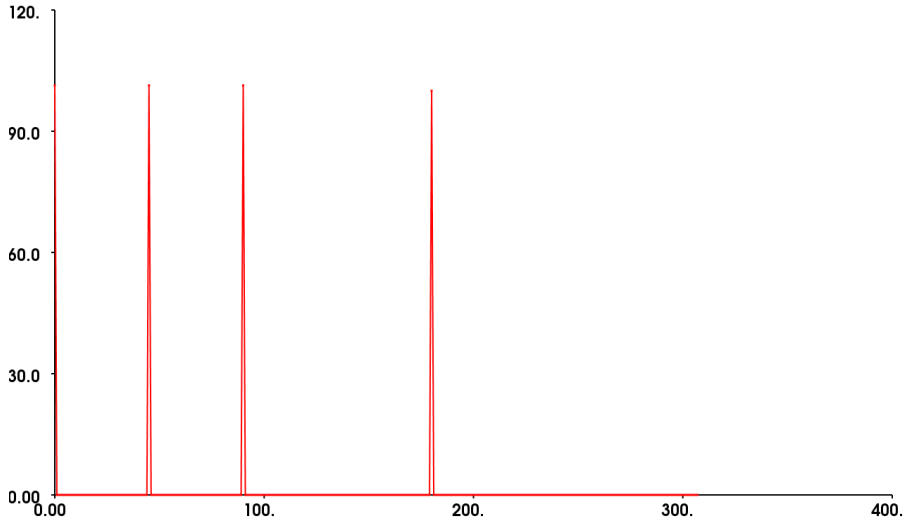


Ilustración 4-11: Histograma del archivo coche_0.pcd con radio de búsqueda de 3 cm.

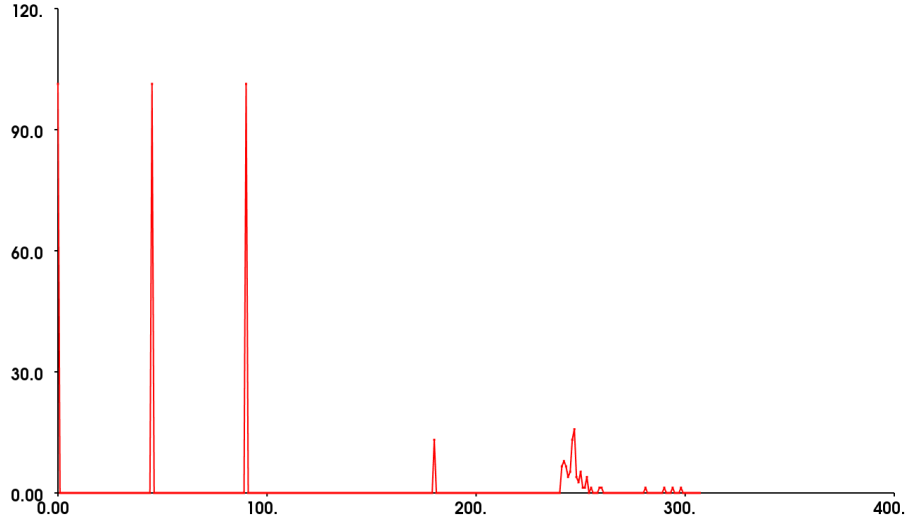


Ilustración 4-12: Histograma del archivo coche_0.pcd con radio de búsqueda de 30 cm.

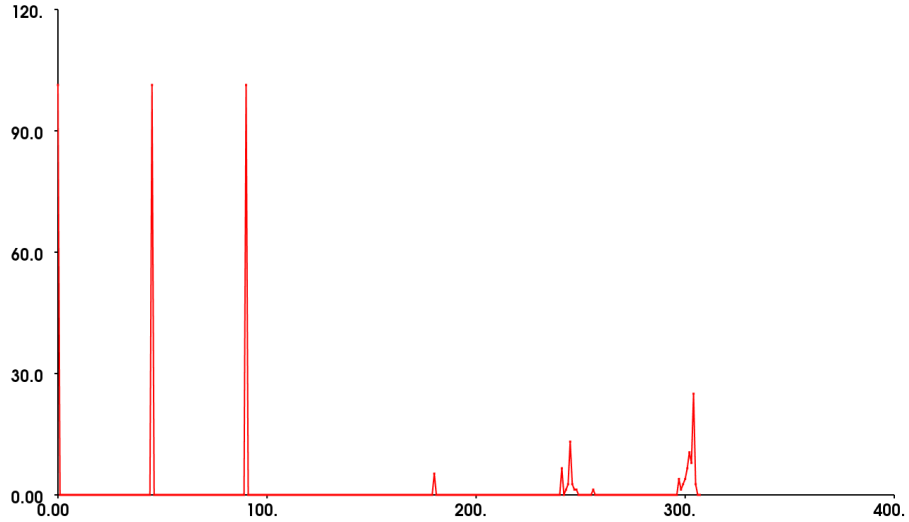


Ilustración 4-13: Histograma del archivo coche_0.pcd con radio de búsqueda de 80 cm.

Se selecciona un radio de búsqueda de 30 cm, de forma que los histogramas de la base de datos del coche y del peatón tengan diferencias (ya que, en el caso del peatón, al tener menor número de puntos, el ruido a la derecha del histograma no aparece):

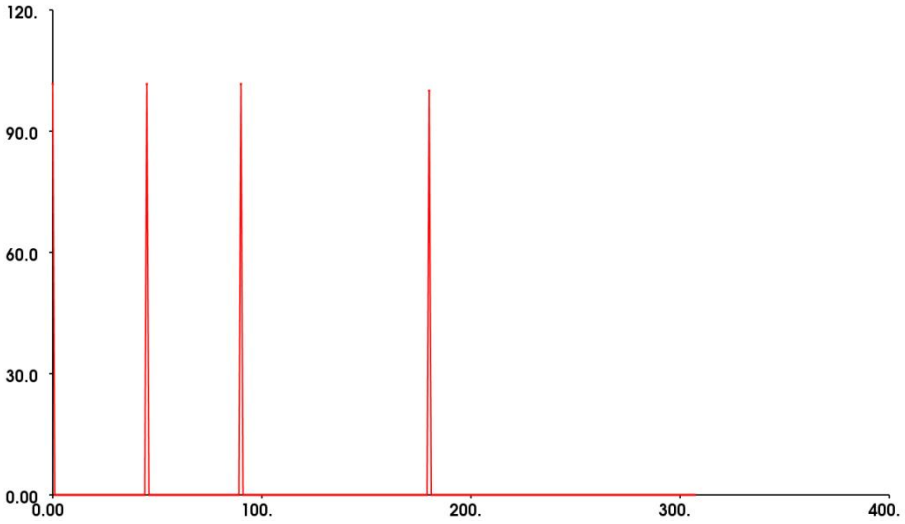


Ilustración 4-14: Histograma del archivo peaton_0.pcd con radio de búsqueda de 30 cm.

Por último, se guarda el descriptor VFH en formato .pcd para trabajar con él más adelante, con el nombre *objeto_i_vfh.pcd,* siendo *objeto* el tipo de obstáculo (coche o peatón) e *i* el ángulo de giro.

4.5.4 Entrenamiento base de datos:

Recognition_6DOF_VFH_training.cpp

Una vez obtenido cada uno de los histogramas de los descriptores VFH de cada uno de los *cluster* que componen la base de datos (cada objeto rotado 360º con saltos de 5º para el sensor LIDAR y 10º para las cámaras), comienza el entrenamiento. En la fase 'training', se generan 3 archivos:

kdtree.idx: estructura organizativa de puntos en el espacio de k dimensiones (en este caso, de 3 dimensiones).

training_data.h5: archivo de extensión .h5 que almacena y organiza una gran cantidad de datos.

training_data.list: se genera una lista con todos los archivos VFH de la base de datos. Para el LIDAR está compuesta por los histogramas de las 144 posiciones posibles, y para la cámara 72 histogramas.

Se crean 2 bases de datos, una para el LIDAR formada por el coche y el peatón rotados cada 5º, y otra para la cámara rotados cada 10º.

4.5.5 Test base de datos:

Recognition_6DOF_VFH_testing.cpp

Generados los 3 archivos descritos anteriormente, se procede a comparar un cluster con la base de datos, de forma que indica qué objetos dentro de la base de datos son similares geométricamente a él. De esta forma, se puede realizar un reconocimiento de objetos.

Se carga el histograma del nuevo objeto a analizar, y la búsqueda de objetos similares se puede realizar de 2 formas:

1.- ***Manual:*** Introduciendo desde el terminal el número de objetos similares se mostrarán y la distancia máxima entre ellos.

2.- ***Automático:*** Se muestran los 6 objetos más parecidos dentro de la base de datos, sin límite de distancia.

Se carga la lista formada anteriormente con todos los objetos de la base de datos y el kd-tree (con los 3 archivos generados).

A continuación, se muestra una comparativa entre la base de datos de los datos procesados por el LIDAR y con las cámaras. Además, se generarán tres versiones de prueba:

Versión 0: se genera la base de datos con los objetos sin escalar y con un radio de búsqueda de 3 cm.

Versión 1: se escala la base de datos y el radio de búsqueda se mantiene en 3 cm.

Versión 2: se mantiene la escala, pero se aumenta el radio de búsqueda a 30 cm.

De esta forma, se mostrará también una comparativa variando el radio de búsqueda del descriptor VFH, con el modo de testeo automático.

4.5.5.1 Sin escalar prueba:

En primer lugar, se realiza la fase de testeo con una base de datos sin escalar (con el tamaño con el que son capturados), y con un radio de búsqueda de 3 cm para el descriptor VFH.

4.5.5.1.1 Base de datos VLP-16:

Se analiza un objeto de cada tipo con el sensor de menor resolución.

Ilustración 4-15: Fase testing del objeto coche_0_vfh.pcd en la versión 0.

Se observa que los candidatos son vehículos al igual que el objeto a testear, pero la pose de ellos no es similar a la del objeto.

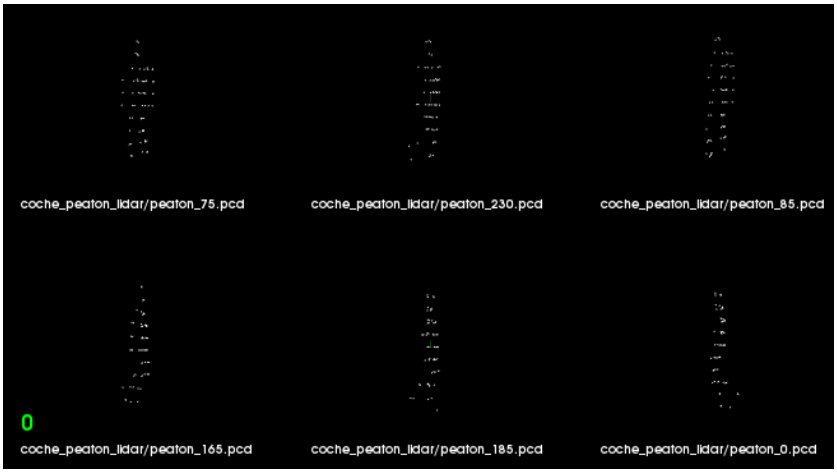


Ilustración 4-16: Fase testing del objeto peaton_0_vfh.pcd en la versión 0.

En el caso del peatón, se detecta el objeto correctamente, pero no la pose. Es un objeto que en todas sus posiciones tiene un aspecto muy similar, por lo que al tener pocos puntos será más complicado identificar la posición.

4.5.5.1.2 Base de datos HDL-64:

Se analiza un objeto de cada tipo con las cámaras, obteniendo nubes de puntos de mayor resolución.

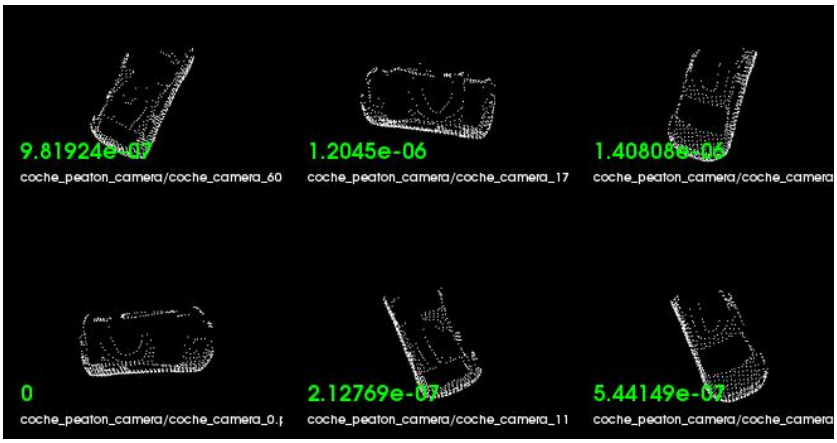


Ilustración 4-17: Fase testing del objeto coche_camera_0_vfh.pcd en la versión 0.

Ilustración 4-18: Fase testing del objeto peaton_camera_0_vfh.pcd en la versión 0.

Los obstáculos se detectan correctamente pero las posiciones siguen sin ser correctas.

4.5.5.2 Radio de búsqueda de 3 cm - escalado:

A continuación, se realiza la fase de testing con una base de datos escalada a 5 metros en altura, y cuyos descriptores tienen un radio de búsqueda entre puntos de 3 cm.

4.5.5.2.1 Base de datos VLP-16:

Se analizan 2 objetos de cada tipo, obteniendo los siguientes resultados:

Ilustración 4-19: Fase testing del objeto coche_0_vfh.pcd en la versión 1.

El objeto analizado se encuentra en la base de datos, pero la orientación de los objetos más similares no se parece a la de dicho objeto.

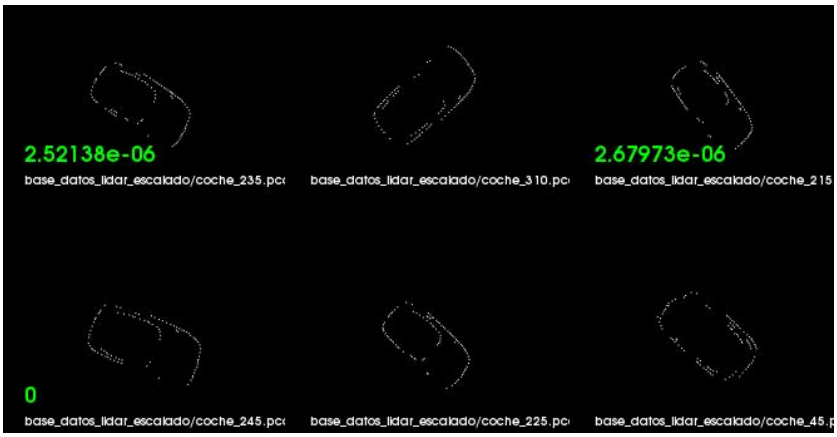


Ilustración 4-20: Fase testing del objeto coche_225_vfh.pcd en la versión 1.

Se obtienen orientaciones más similares a las del objeto analizado que en el caso de la versión 0.

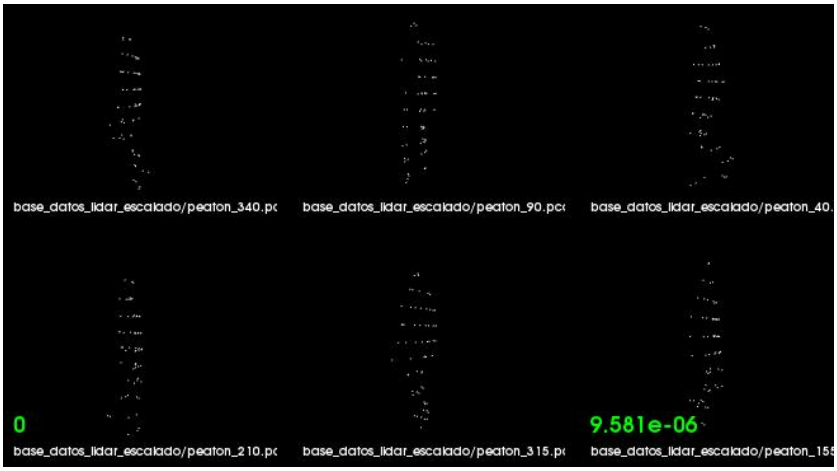


Ilustración 4-21: Fase testing del objeto peaton_210_vfh.pcd en la versión 1.

Ilustración 4-22: Fase testing del objeto peaton_295_vfh.pcd en la versión 1.

4.5.5.2.2 Base de datos HDL-64:

Ilustración 4-23: Fase testing del objeto coche_camera_90_vfh.pcd en la versión 1.

Para este caso, se detecta que el objeto analizado está en la base de datos, y los 5 objetos más parecidos tienen una orientación muy similar (variando en ±10º).

Ilustración 4-24: Fase testing del objeto coche_camera_180_vfh.pcd en la versión 1.

En este caso, obtiene mayor similitud en la orientación que en el caso del LIDAR, pero en 3 de los casos, hay una diferencia de ángulo de aproximadamente 90º.

4.5.5.3 Radio de búsqueda de 30 cm:

Con la misma base de datos del apartado anterior, se calculan los descriptores de cada objeto con un radio de búsqueda ampliado a 30, ya que se observó que en los histogramas existen diferencias entre cada tipo de objeto.

4.5.5.3.1 Base de datos VLP-16:

Se seleccionan dos objetos de cada tipo, y se compararán con la base de datos del LIDAR.

Ilustración 4-25: Fase testing del objeto coche_45_vfh.pcd

El objeto más similar es el que tenga el número más cercano al 0, por lo que el sistema reconoce que el objeto que se está analizando está en la base de datos (identifica que es *coche_45.pcd*). Se observa que los 6 objetos más similares son coches (por lo que no lo confunde con un peatón) y que además todos tienen un ángulo de giro similar sobre el eje Z, en torno a ±n·45º.

Ilustración 4-26: Fase testing del objeto coche_180_vfh.pcd

Al igual que en el caso anterior, se obtiene que el objeto está en la base de datos. Los otros 5 objetos restantes son un coche y además tienen un ángulo de giro similar: siendo el ángulo de giro de 180º, reconoce como similares los que están girados a 170 y 190º.

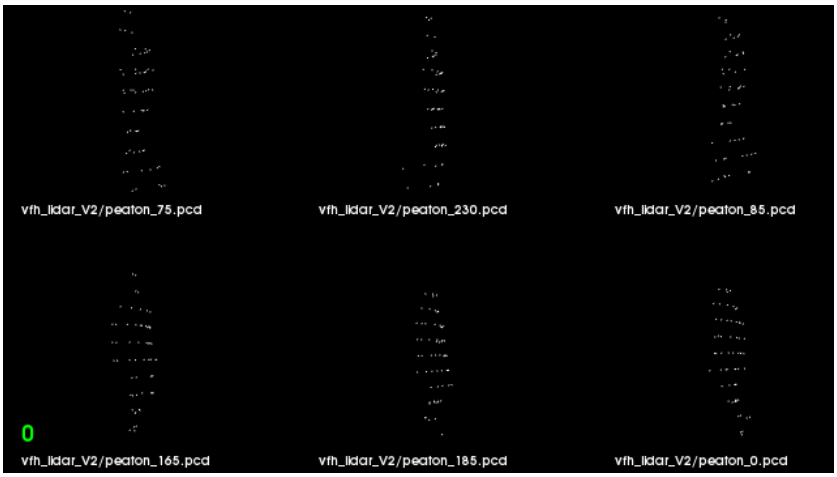


Ilustración 4-27: Fase testing del objeto peaton_0_vfh.pcd

Para este caso, detecta que los objetos más similares también son peatones. También detecta que el objeto que ha sido analizado se encuentra dentro de la base de datos. Para el caso del peatón, el ángulo de giro es más diferente ya que en todas las posiciones la pose es muy similar (a diferencia el coche que cambian las proporciones según la vista).

Ilustración 4-28: Fase testing del objeto peaton_135_vfh.pcd

Ocurre como en el ejemplo anterior, todos los objetos similares son peatones.

4.5.5.3.2 Base de datos HDL-64:

A continuación, se toman 2 objetos capturados por las cámaras y se comparan con la base de datos:

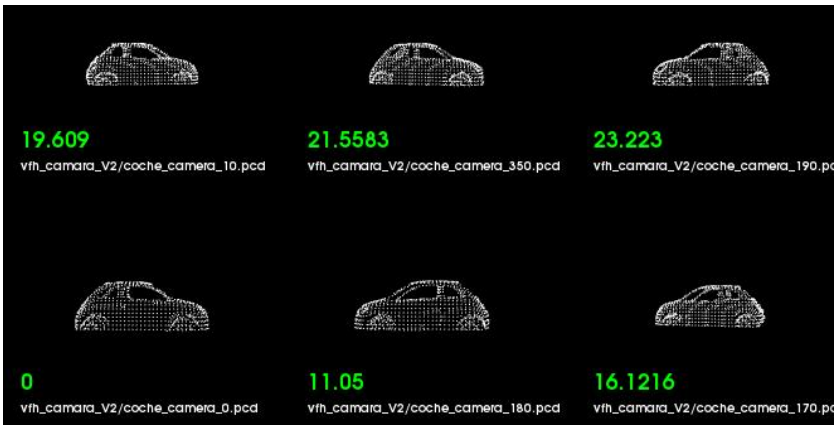


Ilustración 4-29: Fase testing del objeto coche_camera_0_vfh.pcd

Los objetos más similares son vehículos, encontrándose entre ellos el propio objeto analizado. Se observa que el ángulo de rotación es de ±10º con respecto a la posición 0 o con respecto a la posición 180º, que será la más similar ya que el vehículo se encuentra de lado.

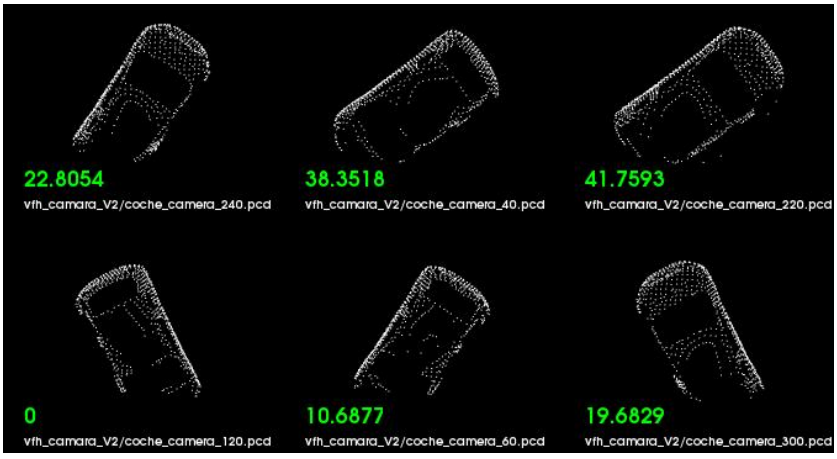


Ilustración 4-30: Fase testing del objeto coche_camera_120_vfh.pcd

Al igual que en el caso de 0º, todos los objetos similares son vehículos, y tienen un ángulo de rotación similar al analizado.

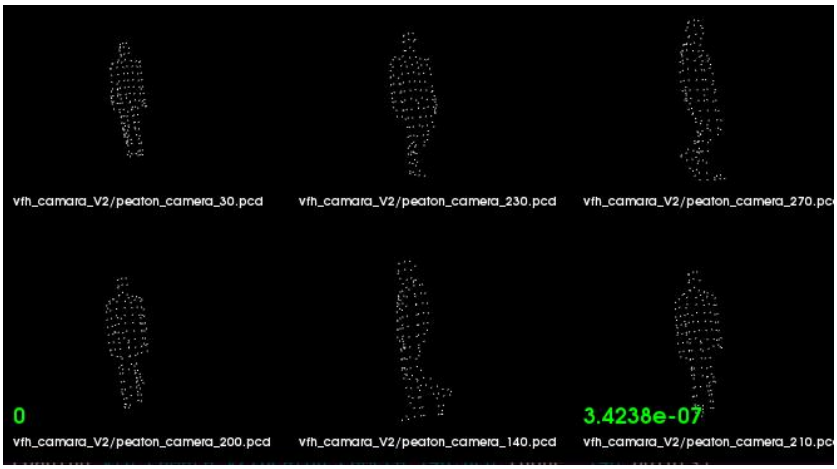


Ilustración 4-31: Fase testing del objeto peaton_camera_200_vfh.pcd

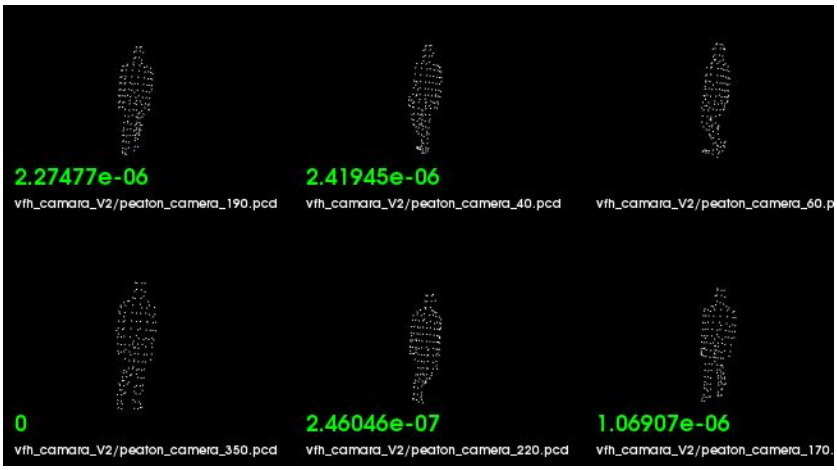


Ilustración 4-32: Fase testing del objeto peaton_camera_350_vfh.pcd

En el caso del peatón, también se obtiene que todos los objetos similares son peatones.

4.5.6 Resultados:

En el primer caso (sin escalar), tanto con los datos del LIDAR como con los de la cámara, se identifica correctamente el objeto, pero la estimación de la pose no la realiza correctamente. Se observa que en el caso de la cámara, la identificación de la pose es un poco más precisa, pero solo detecta un objeto con pose similar.

A continuación, se escalan las bases de datos. La identificación de los objetos en ambos casos continua siendo correcta. En el caso del LIDAR, la precisión de la estimación de la pose sigue siendo baja, pero en el caso de la cámara, mejora la precisión, pero no es correcta al 100%.

En el caso del radio de búsqueda de 30 cm, tanto en la base de datos tomada por el LIDAR (menor resolución), como de la cámara (mayor resolución), se obtiene que al analizar un vehículo el sistema identifica que es un coche, y en caso de analizar un peatón obtiene que es un peatón, por lo que la identificación del objeto es correcta.

La diferencia entre usar el LIDAR o la cámara en este caso se encuentra en la estimación de la pose: en el caso de la cámara, es capaz de detectar con más precisión poses parecidas que en el caso del LIDAR. En comparación con las 2 versiones anteriores, se podría decir que la

precisión de la estimación de la pose es correcta al 100%. Esto es debido a que en los histogramas de cada objeto se obtiene más información en la parte derecha, correspondiente a la parte geométrica del objeto. En la parte izquierda (correspondiente a la orientación del objeto), las diferencias para identificar la pose está en los decimales que acompañan al número, pues al abrir el archivo vfh se observan las similitudes:

Se toma como ejemplo la base de datos del coche realizada con la cámara y un radio de búsqueda de 30 cm, girando desde la posición del reposo (0º) cada 90º. Los valores correspondientes en el histograma a la parte de la orientación del objeto son los siguientes:

- Para el caso de 0º, el valor es de 100.14304
- Para el caso de 90º, el valor es de 100.2371
- Para el caso de 180º, el valor es de 100.13441
- Para el caso de 270º, el valor es de 100.20158

La posición más similar (aparte de comparar con los ángulos cercanos en ±10º), sería sumar al ángulo 180º. Se observa que en el caso de 0º, la cifra es muy similar a la del objeto girado 180º, sin embargo, es muy diferente a la del caso de 90º. En el caso de 90º y 270º (variando en 0.03). Estas similitudes, añadidas a la amplia información de la forma geométrica del objeto, hacen que en reconocimiento sea más preciso.

4.6 RECONOCIMIENTO DE OBJETOS

Se generará una escena con un peatón y otra con un coche (ambos sin movimiento), a la misma distancia que se encontraban cuando se creó la base de datos.

Para ello, se empleará un algoritmo que abre una escena, separa los objetos en clusters individuales y calcula el histograma de cada cluster. Este histograma es comparado con la base de datos correspondiente, y por pantalla se muestran los 16 objetos de la base de datos más parecidos al objeto de la escena.

4.6.1 Base de datos LIDAR:

Dentro de las diferentes versiones, se observa que al modificar los parámetros el reconocimiento va mejorando.

4.6.1.1 Reconocimiento del coche:

```
> The closest 16 neighbors are:
    0 - coche_peaton_lidar/coche_75_vfh.pcd (53) with a distance of: 0.000000
    1 - coche_peaton_lidar/coche_185_vfh.pcd (63) with a distance of: 0.000000
    2 - coche_peaton_lidar/peaton_350_vfh.pcd (65) with a distance of: 0.000000
    3 - coche_peaton_lidar/peaton_220_vfh.pcd (128) with a distance of: 0.000000
    4 - coche_peaton_lidar/coche_120_vfh.pcd (46) with a distance of: 0.000006
    5 - coche_peaton_lidar/peaton_170_vfh.pcd (50) with a distance of: 0.000006
    6 - coche_peaton_lidar/peaton_15_vfh.pcd (99) with a distance of: 0.000006
    7 - coche_peaton_lidar/coche_295_vfh.pcd (107) with a distance of: 0.000006
    8 - coche_peaton_lidar/coche_5_vfh.pcd (26) with a distance of: 0.000007
    9 - coche_peaton_lidar/coche_20_vfh.pcd (66) with a distance of: 0.000007
    10 - coche_peaton_lidar/peaton_345_vfh.pcd (136) with a distance of: 0.000007
    11 - coche_peaton_lidar/coche_255_vfh.pcd (132) with a distance of: 0.000025
    12 - coche_peaton_lidar/peaton_305_vfh.pcd (44) with a distance of: 0.000028
    13 - coche_peaton_lidar/coche_290_vfh.pcd (81) with a distance of: 0.000028
    14 - coche_peaton_lidar/peaton_160_vfh.pcd (83) with a distance of: 0.000028
    15 - coche_peaton_lidar/coche_125_vfh.pcd (48) with a distance of: 0.000054
El elemento más parecido a este objeto es coche_peaton_lidar/coche_75_vfh.pcd
```

Ilustración 4-33: Lista de objetos similares al coche en la versión 0 con VLP16.

Para el caso del reconocimiento de un coche a 9 metros de distancia, con la base de datos versión 0 (objetos sin escalar y con radio de búsqueda de 3 cm), reconoce que el objeto más similar es un coche, pero la tercera opción similar es un peatón. De los 16 objetos más similares, 9 son vehículos y por lo tanto correctos, pero hay 7 que son peatones.

```
> The closest 16 neighbors are:
    0 - base_datos_lidar_escalado/coche_180_vfh.pcd (113) with a distance of: 0.000000
    1 - base_datos_lidar_escalado/coche_10_vfh.pcd (123) with a distance of: 0.000171
    2 - base_datos_lidar_escalado/coche_100_vfh.pcd (101) with a distance of: 0.000369
    3 - base_datos_lidar_escalado/coche_95_vfh.pcd (28) with a distance of: 0.000629
    4 - base_datos_lidar_escalado/coche_30_vfh.pcd (111) with a distance of: 0.000919
    5 - base_datos_lidar_escalado/coche_350_vfh.pcd (11) with a distance of: 0.000941
    6 - base_datos_lidar_escalado/coche_190_vfh.pcd (133) with a distance of: 0.000941
    7 - base_datos_lidar_escalado/coche_165_vfh.pcd (142) with a distance of: 0.001697
    8 - base_datos_lidar_escalado/coche_170_vfh.pcd (15) with a distance of: 0.002591
    9 - base_datos_lidar_escalado/peaton_125_vfh.pcd (30) with a distance of: 0.002591
    10 - base_datos_lidar_escalado/peaton_100_vfh.pcd (126) with a distance of: 0.00259
    11 - base_datos_lidar_escalado/peaton_245_vfh.pcd (10) with a distance of: 0.003077
    12 - base_datos_lidar_escalado/peaton_105_vfh.pcd (56) with a distance of: 0.003077
    13 - base_datos_lidar_escalado/peaton_115_vfh.pcd (119) with a distance of: 0.00307
    14 - base_datos_lidar_escalado/coche_15_vfh.pcd (29) with a distance of: 0.003585
    15 - base_datos_lidar_escalado/coche_270_vfh.pcd (60) with a distance of: 0.003585
El elemento más parecido a este objeto es base_datos_lidar_escalado/coche_180_vfh.pcd
```

Ilustración 4-34: Lista de objetos similares al coche en la versión 1 con VLP16.

Con la base de datos versión 1, escalando todos los objetos de la base de datos para que la altura sea de 5 metros, se obtiene que el objeto reconocido es un coche, pero el décimo más similar es un peatón. Se mejora la probabilidad de acierto: de los 16 objetos más similares en la base de datos, 11 son coches y solo 5 peatones.

```
> The closest 16 neighbors are:
    0 - vfh_lidar_V2/coche_330_vfh.pcd (80) with a distance of: 18.838572
    1 - vfh_lidar_V2/coche_350_vfh.pcd (11) with a distance of: 21.256866
    2 - vfh_lidar_V2/coche_215_vfh.pcd (5) with a distance of: 32.636772
    3 - vfh_lidar_V2/coche_325_vfh.pcd (41) with a distance of: 34.128132
    4 - vfh_lidar_V2/coche_340_vfh.pcd (85) with a distance of: 34.633316
    5 - vfh_lidar_V2/coche_315_vfh.pcd (93) with a distance of: 36.296432
    6 - vfh_lidar_V2/coche_20_vfh.pcd (66) with a distance of: 36.380169
    7 - vfh_lidar_V2/coche_175_vfh.pcd (92) with a distance of: 36.994785
    8 - vfh_lidar_V2/coche_310_vfh.pcd (137) with a distance of: 37.065998
    9 - vfh_lidar_V2/coche_195_vfh.pcd (77) with a distance of: 38.749268
    10 - vfh_lidar_V2/coche_0_vfh.pcd (79) with a distance of: 39.963501
    11 - vfh_lidar_V2/coche_345_vfh.pcd (27) with a distance of: 40.130936
    12 - vfh_lidar_V2/coche_235_vfh.pcd (23) with a distance of: 40.791992
    13 - vfh_lidar_V2/coche_140_vfh.pcd (42) with a distance of: 41.450413
    14 - vfh_lidar_V2/coche_25_vfh.pcd (9) with a distance of: 41.567600
    15 - vfh_lidar_V2/coche_35_vfh.pcd (0) with a distance of: 43.637817
El elemento más parecido a este objeto es vfh_lidar_V2/coche_330_vfh.pcd
```

Ilustración 4-35: Lista de objetos similares al coche en la versión 2 con VLP16.

Con la versión 2 se aumenta el radio de búsqueda entre puntos de un cluster a 30 cm. Se obtiene que los 16 objetos más similares son coches, por lo que la probabilidad de acierto es de un 100%.

4.6.1.2 Reconocimiento del peatón:

```
> ELEMENTO ACTUAL ROJO

 > The closest 16 neighbors are:
    0 - coche_peaton_lidar/peaton_210_vfh.pcd (95) with a distance of: 0.000000
    1 - coche_peaton_lidar/peaton_315_vfh.pcd (120) with a distance of: 0.000000
    2 - coche_peaton_lidar/peaton_155_vfh.pcd (4) with a distance of: 0.000010
    3 - coche_peaton_lidar/peaton_90_vfh.pcd (49) with a distance of: 0.000010
    4 - coche_peaton_lidar/peaton_40_vfh.pcd (96) with a distance of: 0.000010
    5 - coche_peaton_lidar/coche_80_vfh.pcd (97) with a distance of: 0.000010
    6 - coche_peaton_lidar/peaton_80_vfh.pcd (106) with a distance of: 0.000010
    7 - coche_peaton_lidar/peaton_50_vfh.pcd (110) with a distance of: 0.000010
    8 - coche_peaton_lidar/peaton_140_vfh.pcd (22) with a distance of: 0.000010
    9 - coche_peaton_lidar/peaton_290_vfh.pcd (51) with a distance of: 0.000010
    10 - coche_peaton_lidar/coche_335_vfh.pcd (76) with a distance of: 0.000010
    11 - coche_peaton_lidar/peaton_335_vfh.pcd (139) with a distance of: 0.000010
    12 - coche_peaton_lidar/peaton_325_vfh.pcd (16) with a distance of: 0.000037
    13 - coche_peaton_lidar/coche_260_vfh.pcd (18) with a distance of: 0.000037
    14 - coche_peaton_lidar/peaton_60_vfh.pcd (34) with a distance of: 0.000037
    15 - coche_peaton_lidar/peaton_225_vfh.pcd (52) with a distance of: 0.000037
El elemento más parecido a este objeto es coche_peaton_lidar/peaton_210_vfh.pcd
```

Ilustración 4-36: Lista de objetos similares al peatón en la versión 0 con VLP16.

Para el caso del peatón, en la versión 0 el candidato más similar es el peatón, pero de 16 candidatos, 13 son peatones y los 3 restantes coches.

```
 > The closest 16 neighbors are:
    0 - base_datos_lidar_escalado/peaton_155_vfh.pcd (4) with a distance of: 0.000000
    1 - base_datos_lidar_escalado/peaton_340_vfh.pcd (36) with a distance of: 0.000000
    2 - base_datos_lidar_escalado/peaton_90_vfh.pcd (49) with a distance of: 0.000000
    3 - base_datos_lidar_escalado/peaton_40_vfh.pcd (96) with a distance of: 0.000000
    4 - base_datos_lidar_escalado/coche_80_vfh.pcd (97) with a distance of: 0.000000
    5 - base_datos_lidar_escalado/peaton_80_vfh.pcd (106) with a distance of: 0.000000
    6 - base_datos_lidar_escalado/peaton_50_vfh.pcd (110) with a distance of: 0.000000
    7 - base_datos_lidar_escalado/peaton_325_vfh.pcd (16) with a distance of: 0.000009
    8 - base_datos_lidar_escalado/coche_260_vfh.pcd (18) with a distance of: 0.000009
    9 - base_datos_lidar_escalado/peaton_60_vfh.pcd (34) with a distance of: 0.000009
    10 - base_datos_lidar_escalado/peaton_225_vfh.pcd (52) with a distance of: 0.000009
    11 - base_datos_lidar_escalado/peaton_5_vfh.pcd (58) with a distance of: 0.000009
    12 - base_datos_lidar_escalado/peaton_300_vfh.pcd (87) with a distance of: 0.000009
    13 - base_datos_lidar_escalado/peaton_10_vfh.pcd (102) with a distance of: 0.000009
    14 - base_datos_lidar_escalado/peaton_310_vfh.pcd (130) with a distance of: 0.000009
    15 - base_datos_lidar_escalado/peaton_355_vfh.pcd (131) with a distance of: 0.000009
El elemento más parecido a este objeto es base_datos_lidar_escalado/peaton_155_vfh.pcd
```

Ilustración 4-37: Lista de objetos similares al peatón en la versión 1 con VLP16.

Para la versión 1 con el peatón, el número de candidatos erróneos se reduce a 2, por lo que existen 14 de 16 candidatos correctos.

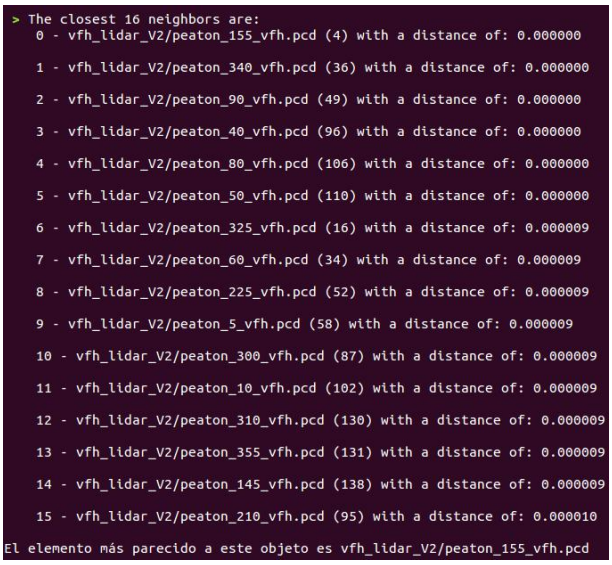


Ilustración 4-38: Lista de objetos similares al peatón en la versión 2 con VLP16.

Por último, en la versión 2, todos los candidatos similares son peatones, por lo que existe un 100% de probabilidades de acierto.

En el caso del LIDAR, existe más probabilidad de fallo en la detección de vehículos que de peatones, pero variando el radio de detección para el descriptor VFH se consigue corregir.

4.6.2 Base de datos cámaras:

De la misma forma, se crean 2 escenas, una con un vehículo y otra con un peatón sin movimiento, a la misma distancia que los objetos de la base de datos. Se probarán las 3 versiones y se analizará si el reconocimiento es correcto.

4.6.2.1 Reconocimiento del coche:

```
> The closest 16 neighbors are:
    0 - coche_peaton_camera/coche_camera_90_vfh.pcd (40) with a distance of: 0.000001
    1 - coche_peaton_camera/coche_camera_270_vfh.pcd (41) with a distance of: 0.000030
    2 - coche_peaton_camera/coche_camera_100_vfh.pcd (13) with a distance of: 0.000100
    3 - coche_peaton_camera/coche_camera_80_vfh.pcd (49) with a distance of: 0.000122
    4 - coche_peaton_camera/coche_camera_280_vfh.pcd (56) with a distance of: 0.000123
    5 - coche_peaton_camera/coche_camera_110_vfh.pcd (36) with a distance of: 0.000149
    6 - coche_peaton_camera/coche_camera_0_vfh.pcd (52) with a distance of: 0.000160
    7 - coche_peaton_camera/coche_camera_290_vfh.pcd (11) with a distance of: 0.000179
    8 - coche_peaton_camera/coche_camera_60_vfh.pcd (66) with a distance of: 0.000186
    9 - coche_peaton_camera/coche_camera_170_vfh.pcd (20) with a distance of: 0.000189
    10 - coche_peaton_camera/coche_camera_250_vfh.pcd (1) with a distance of: 0.000192
    11 - coche_peaton_camera/coche_camera_300_vfh.pcd (61) with a distance of: 0.000201
    12 - coche_peaton_camera/coche_camera_190_vfh.pcd (44) with a distance of: 0.000207
    13 - coche_peaton_camera/coche_camera_120_vfh.pcd (31) with a distance of: 0.000213
    14 - coche_peaton_camera/coche_camera_40_vfh.pcd (18) with a distance of: 0.000216
    15 - coche_peaton_camera/coche_camera_240_vfh.pcd (46) with a distance of: 0.000221
El elemento más parecido a este objeto es coche_peaton_camera/coche_camera_90_vfh.pcd
```

Ilustración 4-39: Lista de objetos similares al coche en la versión 0 con HDL64.

```
> The closest 16 neighbors are:
    0 - base_datos_camara_escalado/coche_camera_90_vfh.pcd (40) with a distance of: 0.000004
    1 - base_datos_camara_escalado/coche_camera_270_vfh.pcd (41) with a distance of: 0.000040
    2 - base_datos_camara_escalado/coche_camera_100_vfh.pcd (13) with a distance of: 0.000118
    3 - base_datos_camara_escalado/coche_camera_80_vfh.pcd (49) with a distance of: 0.000142
    4 - base_datos_camara_escalado/coche_camera_280_vfh.pcd (56) with a distance of: 0.000143
    5 - base_datos_camara_escalado/coche_camera_260_vfh.pcd (48) with a distance of: 0.000148
    6 - base_datos_camara_escalado/coche_camera_110_vfh.pcd (36) with a distance of: 0.000171
    7 - base_datos_camara_escalado/coche_camera_0_vfh.pcd (52) with a distance of: 0.000183
    8 - base_datos_camara_escalado/coche_camera_70_vfh.pcd (15) with a distance of: 0.000194
    9 - base_datos_camara_escalado/coche_camera_290_vfh.pcd (11) with a distance of: 0.000203
    10 - base_datos_camara_escalado/coche_camera_60_vfh.pcd (66) with a distance of: 0.000211
    11 - base_datos_camara_escalado/coche_camera_180_vfh.pcd (54) with a distance of: 0.000213
    12 - base_datos_camara_escalado/coche_camera_170_vfh.pcd (20) with a distance of: 0.000214
    13 - base_datos_camara_escalado/coche_camera_250_vfh.pcd (1) with a distance of: 0.000216
    14 - base_datos_camara_escalado/coche_camera_10_vfh.pcd (9) with a distance of: 0.000218
    15 - base_datos_camara_escalado/coche_camera_300_vfh.pcd (61) with a distance of: 0.000226
El elemento más parecido a este objeto es base_datos_camara_escalado/coche_camera_90_vfh.pcd
```

Ilustración 4-40: Lista de objetos similares al coche en la versión 1 con HDL64.

```
> The closest 16 neighbors are:
    0 - vfh_camara_V2/coche_camera_340_vfh.pcd (33) with a distance of: 31.576109
    1 - vfh_camara_V2/coche_camera_160_vfh.pcd (3) with a distance of: 32.376900
    2 - vfh_camara_V2/coche_camera_200_vfh.pcd (42) with a distance of: 32.824478
    3 - vfh_camara_V2/coche_camera_180_vfh.pcd (54) with a distance of: 34.765121
    4 - vfh_camara_V2/coche_camera_20_vfh.pcd (35) with a distance of: 35.655834
    5 - vfh_camara_V2/coche_camera_0_vfh.pcd (52) with a distance of: 35.882000
    6 - vfh_camara_V2/coche_camera_350_vfh.pcd (27) with a distance of: 38.716816
    7 - vfh_camara_V2/coche_camera_170_vfh.pcd (20) with a distance of: 39.624065
    8 - vfh_camara_V2/coche_camera_190_vfh.pcd (44) with a distance of: 40.164661
    9 - vfh_camara_V2/coche_camera_10_vfh.pcd (9) with a distance of: 43.155487
    10 - vfh_camara_V2/coche_camera_90_vfh.pcd (40) with a distance of: 56.154396
    11 - vfh_camara_V2/coche_camera_100_vfh.pcd (13) with a distance of: 60.476658
    12 - vfh_camara_V2/coche_camera_330_vfh.pcd (34) with a distance of: 60.942314
    13 - vfh_camara_V2/coche_camera_80_vfh.pcd (49) with a distance of: 64.069153
    14 - vfh_camara_V2/coche_camera_270_vfh.pcd (41) with a distance of: 65.636688
    15 - vfh_camara_V2/coche_camera_70_vfh.pcd (15) with a distance of: 65.757248
El elemento más parecido a este objeto es vfh_camara_V2/coche_camera_340_vfh.pcd
```

Ilustración 4-41: Lista de objetos similares al coche en la versión 2 con HDL64.

A diferencia del caso del coche detectado mediante el LIDAR, en los tres casos de la cámara se consigue detectar perfectamente el vehículo y, por tanto, reconocerlo. Los 16 posibles candidatos para todos los casos son vehículos. Esto es debido a que la nube de puntos contiene más puntos y por ello, más información.

4.6.2.2 Reconocimiento del peatón:

```
> The closest 16 neighbors are:
    0 - coche_peaton_camera/peaton_camera_50_vfh.pcd (38) with a distance of: 0.000000
    1 - coche_peaton_camera/peaton_camera_310_vfh.pcd (51) with a distance of: 0.000001
    2 - coche_peaton_camera/peaton_camera_290_vfh.pcd (71) with a distance of: 0.000001
    3 - coche_peaton_camera/peaton_camera_320_vfh.pcd (30) with a distance of: 0.000003
    4 - coche_peaton_camera/peaton_camera_200_vfh.pcd (24) with a distance of: 0.000005
    5 - coche_peaton_camera/peaton_camera_140_vfh.pcd (39) with a distance of: 0.000005
    6 - coche_peaton_camera/peaton_camera_210_vfh.pcd (0) with a distance of: 0.000008
    7 - coche_peaton_camera/peaton_camera_30_vfh.pcd (4) with a distance of: 0.000008
    8 - coche_peaton_camera/peaton_camera_230_vfh.pcd (25) with a distance of: 0.000008
    9 - coche_peaton_camera/peaton_camera_270_vfh.pcd (53) with a distance of: 0.000008
    10 - coche_peaton_camera/peaton_camera_100_vfh.pcd (19) with a distance of: 0.000010
    11 - coche_peaton_camera/peaton_camera_70_vfh.pcd (58) with a distance of: 0.000012
    12 - coche_peaton_camera/peaton_camera_130_vfh.pcd (22) with a distance of: 0.000014
    13 - coche_peaton_camera/peaton_camera_340_vfh.pcd (62) with a distance of: 0.000016
    14 - coche_peaton_camera/peaton_camera_40_vfh.pcd (5) with a distance of: 0.000025
    15 - coche_peaton_camera/peaton_camera_280_vfh.pcd (50) with a distance of: 0.000025
El elemento más parecido a este objeto es coche_peaton_camera/peaton_camera_50_vfh.pcd
```

Ilustración 4-42: Lista de objetos similares al peatón en la versión 0 con HDL64.

```
> The closest 16 neighbors are:
    0 - base_datos_camara_escalado/peaton_camera_100_vfh.pcd (19) with a distance of: 0.000000
    1 - base_datos_camara_escalado/peaton_camera_130_vfh.pcd (22) with a distance of: 0.000000
    2 - base_datos_camara_escalado/peaton_camera_320_vfh.pcd (30) with a distance of: 0.000002
    3 - base_datos_camara_escalado/peaton_camera_280_vfh.pcd (50) with a distance of: 0.000004
    4 - base_datos_camara_escalado/peaton_camera_250_vfh.pcd (7) with a distance of: 0.000007
    5 - base_datos_camara_escalado/peaton_camera_260_vfh.pcd (21) with a distance of: 0.000007
    6 - base_datos_camara_escalado/peaton_camera_330_vfh.pcd (8) with a distance of: 0.000010
    7 - base_datos_camara_escalado/peaton_camera_50_vfh.pcd (38) with a distance of: 0.000010
    8 - base_datos_camara_escalado/peaton_camera_110_vfh.pcd (12) with a distance of: 0.000016
    9 - base_datos_camara_escalado/peaton_camera_310_vfh.pcd (51) with a distance of: 0.000018
    10 - base_datos_camara_escalado/peaton_camera_290_vfh.pcd (71) with a distance of: 0.000018
    11 - base_datos_camara_escalado/peaton_camera_200_vfh.pcd (24) with a distance of: 0.000029
    12 - base_datos_camara_escalado/peaton_camera_140_vfh.pcd (39) with a distance of: 0.000029
    13 - base_datos_camara_escalado/peaton_camera_210_vfh.pcd (0) with a distance of: 0.000036
    14 - base_datos_camara_escalado/peaton_camera_30_vfh.pcd (4) with a distance of: 0.000036
    15 - base_datos_camara_escalado/peaton_camera_230_vfh.pcd (25) with a distance of: 0.000036
El elemento más parecido a este objeto es base_datos_camara_escalado/peaton_camera_100_vfh.pcd
```

Ilustración 4-43: Lista de objetos similares al peatón en la versión 1 con HDL64.

```
> The closest 16 neighbors are:
    0 - vfh_camara_V2/peaton_camera_330_vfh.pcd (8) with a distance of: 0.000000
    1 - vfh_camara_V2/peaton_camera_50_vfh.pcd (38) with a distance of: 0.000000
    2 - vfh_camara_V2/peaton_camera_310_vfh.pcd (51) with a distance of: 0.000001
    3 - vfh_camara_V2/peaton_camera_290_vfh.pcd (71) with a distance of: 0.000001
    4 - vfh_camara_V2/peaton_camera_320_vfh.pcd (30) with a distance of: 0.000003
    5 - vfh_camara_V2/peaton_camera_200_vfh.pcd (24) with a distance of: 0.000005
    6 - vfh_camara_V2/peaton_camera_140_vfh.pcd (39) with a distance of: 0.000005
    7 - vfh_camara_V2/peaton_camera_210_vfh.pcd (0) with a distance of: 0.000008
    8 - vfh_camara_V2/peaton_camera_30_vfh.pcd (4) with a distance of: 0.000008
    9 - vfh_camara_V2/peaton_camera_230_vfh.pcd (25) with a distance of: 0.000008
    10 - vfh_camara_V2/peaton_camera_270_vfh.pcd (53) with a distance of: 0.000008
    11 - vfh_camara_V2/peaton_camera_100_vfh.pcd (19) with a distance of: 0.000010
    12 - vfh_camara_V2/peaton_camera_70_vfh.pcd (58) with a distance of: 0.000012
    13 - vfh_camara_V2/peaton_camera_130_vfh.pcd (22) with a distance of: 0.000014
    14 - vfh_camara_V2/peaton_camera_340_vfh.pcd (62) with a distance of: 0.000016
    15 - vfh_camara_V2/peaton_camera_90_vfh.pcd (32) with a distance of: 0.000020
El elemento más parecido a este objeto es vfh_camara_V2/peaton_camera_330_vfh.pcd
```

Ilustración 4-44: Lista de objetos similares al peatón en la versión 2 con HDL64.

Sucede lo mismo con el peatón: los 16 candidatos para los tres casos son peatones.

En el caso de la cámara, la identificación de los objetos es correcta, debido a que la resolución de los datos es mayor. Por ello, en las 3 versiones, se alcanza un porcentaje del 100% de aciertos.

5 TRACKING Y DESCRIPTORES VFH

Una vez generadas las bases de datos y ajustados los parámetros del descriptor VFH, se procede a probar todo el sistema.

Se crea un algoritmo que segmentará una nube de datos de una escena en movimiento, escalará cada cluster de forma individual y lo comparará con la base de datos correspondiente mediante el descriptor VFH.

Además de esto, se añadirá un algoritmo para realizar un seguimiento de los objetos en movimiento, denominado ***Filtro de Kalman***. De esta forma, se evita que cuando un cluster se encuentra en movimiento y cambia de posición, no se detecte como un objeto nuevo, si no que se detecte como el mismo objeto que ha cambiado su posición.

5.1 FILTRO DE KALMAN:

El filtro de Kalman, también denominado Estimador Cuadrático Lineal (LQE), es un estimador de estados discreto mediante la medición de los estados del sistema, siendo capaz de realizar un filtrado óptimo aun con la presencia de ruido. Es un algoritmo recursivo que es capaz de corregirse, y depende únicamente del estado anterior y del estado actual, permitiendo el ahorro de memoria en los sistemas digitales.

Consiste en la observación y recopilación de los datos de un sistema en el tiempo, y de forma estadística, se consigue:

1- Predecir el siguiente estado.

2- Mediante la medida del ruido, es capaz de ajustar la estimación del siguiente estado.

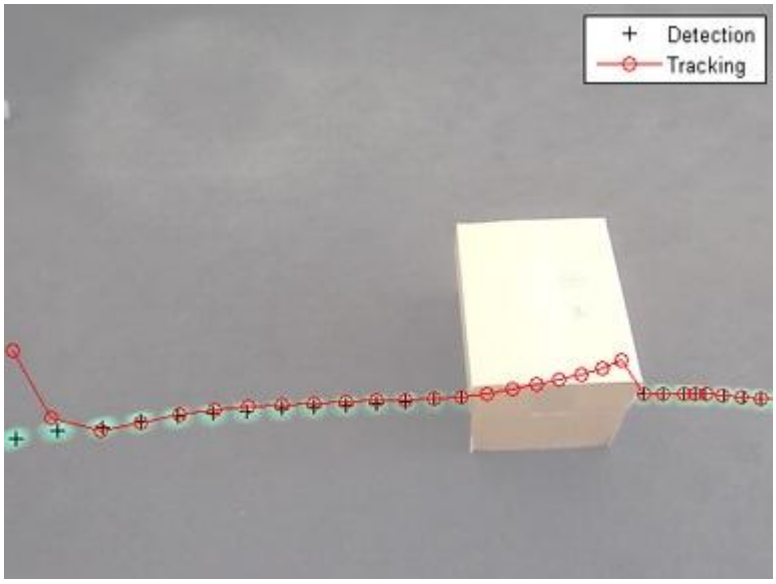


Ilustración 5-1: Trayectoria de un objeto empleado LQE.

En la actualidad, este filtro se utiliza para controlar los sistemas de navegación y visión artificial, de forma que se puede predecir la ubicación de un objeto.

5.2 CLASIFICACIÓN DE OBJETOS

Se genera un algoritmo que recibe una nube de puntos de una escena de VREP con información RGB, por lo que es necesario emplear el archivo launch "**coloring.launch**".

Es necesario suscribirse al topic "***/velodyne_colored_points***" que recibe la nube de tipo XYZRGB.

En el archivo generado, se pueden distinguir dos grupos de funciones:

El primer grupo se encargará de cargar la base de datos del LIDAR para el algoritmo"**publish_object_vlp16_tracking.cpp**" o la base de datos de la cámara para el algoritmo "**publish_object_hdl64e_tracking.cpp**". Se cargan los archivos: kdtree.idx, training_data.h5, training_data.list y se declaran las variables necesarias para emplear el descriptor VFH.

El segundo grupo se encarga de crear el filtro de Kalman: se declaran las variables necesarias, se inicializan las matrices necesarias para calcular las predicciones, y se recalculan en función de los errores en la medida.

Definidos estos parámetros, se procede a trabajar con la escena. En primer lugar, se deberá segmentar la escena y dividirla en clusters individuales. Para ello, se emplea una función denominada `kalmanfilter_node::segmentation_filter` que combina el método de extracción Euclídeo con una segmentación por colores.

Una forma rápida de realizar el reconocimiento de objetos es mediante la separación por colores: como ocurre con las imágenes CNN, tras haber entrenado con una base de datos, se obtiene la imagen resultante de una escena dibujando cada objeto de un color.

La función `kalmanfilter_node::segmentation_filter` recibe la nube completa, unos valores máximos y mínimos de los colores RGB (rojo, verde y azul) y los parámetros para el algoritmo de extracción euclídeo (tolerancia, número mínimo y máximo de puntos por cluster). De esta forma, conociendo el color de cada objeto y aproximadamente el rango de tamaño de puntos, se puede separar cada objeto de forma individual.

Segmentando y eliminando los objetos de colores que no interesan (como por ejemplo, los árboles), se consigue reducir tiempo de cómputo y analizar solo los objetos de interés.

Separados los objetos en clusters, se calcula su centroide y sus longitudes en los ejes XYZ. Para trabajar con el descriptor VFH se necesita eliminar el color de la nube de puntos, por lo que se usa la librería de conversiones de PCL. Debido a que no existe ninguna función que transforme del tipo PointXYZRGB a PointXYZ, se crea la siguiente función:

```
  void PointCloudXYZRGBtoXYZ (pcl::PointCloud<pcl::PointXYZRGB>& input_cloud,
pcl::PointCloud<pcl::PointXYZ>& output_cloud)

   {

     output_cloud.width    = input_cloud.width;

     output_cloud.height   = input_cloud.height;

     for (size_t i = 0; i < input_cloud.points.size (); i++)

     {
```

```
        pcl::PointXYZ p;
        PointXYZRGBtoXYZ (input_cloud.points[i], p);
        output_cloud.points.push_back (p);
      }
    }
```

Esta función recibe una nube de puntos del tipo PointXYZRGB (input_cloud) y devuelve una nueva nube de puntos de tipo PointXYZ (output_cloud). Se obliga a que la nueva nube de puntos tenga las mismas dimensiones que la nube de puntos original, y mediante un bucle for se recorren todos los puntos de la misma, copiándolos en la nueva nube con la siguiente función:

```
void PointXYZRGBtoXYZ (const pcl::PointXYZRGB& in,
                       pcl::PointXYZ&          out)
  {
    out.x = in.x; out.y = in.y; out.z = in.z;
}
```

Una vez eliminado el color, se escala la nube haciendo que tenga una longitud de 5 en el eje Z (como los objetos de la base de datos) y se calcula el descriptor VFH de cada cluster con un radio de búsqueda de 30 cm. De este modo, se compara cada cluster con la base de datos y se identifica qué tipo de objeto es.

Segmentados e identificados los objetos, se procede a calcular la predicción de la trayectoria de cada objeto con el fin de que un mismo obstáculo no sea detectado como varios objetos. Para ello se emplea el filtro de Kalman, y en el visualizador se dibuja un cubo para el objeto, otro para la predicción de la siguiente posición del objeto, la trayectoria actual del objeto y la predicción de la trayectoria del objeto en 10 segundos.

Además, se publicará un topic que indicará mediante un marker de tipo point la posición actual del objeto, de forma que se indica cuántos objetos hay en la escena para visualizarlo en RViz.

Para visualizar el identificador de cada objeto, se crea una variable de tipo string denominada ss_text, donde se almacena el nombre del objeto más similar:

```
ss_text << models.at (k_indices[0][0]).first.c_str () <<"\n";
```

La posición en el visualizador se define mediante la creación de un punto en 3D de tipo pcl::PointXYZ, donde la coordenada X será el valor del centroide en el eje X, la coordenada Y la altura del objeto, y la coordenada Z el valor del centroide en el eje Z.

Para visualizarlo en RViz, se crea un marker de tipo text (`visualization_msgs::Marker::TEXT_VIEW_FACING`), cuya posición se define con las mismas coordenadas que en el visualizador 3D, de la siguiente forma:

```
text.pose.position.x = objeto.centroid_x[0];
text.pose.position.y = objeto.y_max;
```

```
text.pose.position.z = objeto.centroid_z[0];
text.text = (ss_text.str()); //Texto a mostrar
```

5.3 CASOS DE USO

Con el simulador VREP, se generarán diferentes escenas de forma que se simulen situaciones reales en las que se podrá encontrar el vehículo con diferentes obstáculos.

A continuación, se lanzará el algoritmo correspondiente (publish_object_sensor_tracking.cpp), que abrirá la base de datos de cada sensor, y se realizará el reconocimiento con ambos casos, comparando las diferentes técnicas.

Se trabajará con nubes de puntos de tipo RGB. La información del color es proporcionada por las cámaras, que se encuentran en la parte posterior del vehículo (una a la izquierda y otra a la derecha), por lo que se reduce el entorno de análisis en comparación con el caso del LIDAR.

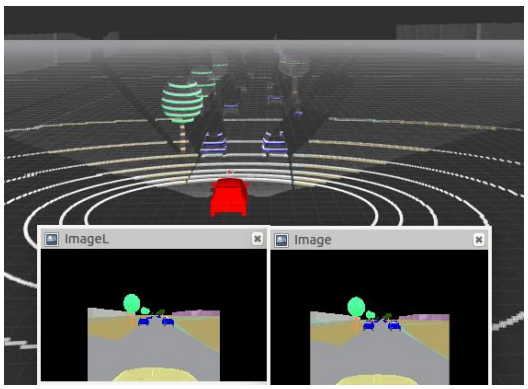


Ilustración 5-2: Segmentación de la nube RGB.

5.3.1 Caso 1: Persecución de un vehículo

El primer caso de uso que se analizará es el reconocimiento de un vehículo en persecución.

Para ello, se crea una escena en el entorno politécnico, donde el vehículo sensorizado y el vehículo en movimiento se mueven en la misma dirección y sentido. Se encuentran separados por una distancia de aproximadamente 6,4 metros, y avanzan a una velocidad de 5 m/s (18 km/h). En el simulador, se ve de la siguiente forma:

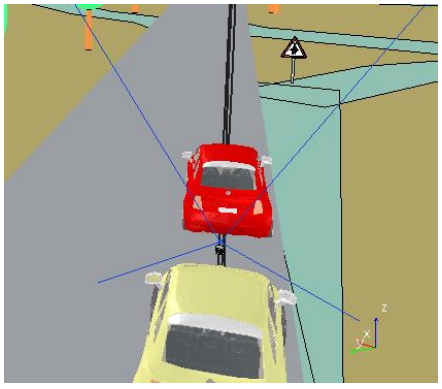


Ilustración 5-3: Caso 1 en V-REP

Debido a que el vehículo a perseguir siempre va delante del sensorizado, en ningún momento desaparecerá del rango de alcance de los sensores.

5.3.1.1 Caso 1: Resultados con VLP-16

Se almacenan un total de 60 resultados de reconocimiento de esta simulación. El objeto debería ser detectado de la misma forma, debido a que ambos avanzan en la misma dirección y sentido con la misma velocidad. Se observa que no es así: el 10% de las muestras son erróneas mientras que el 90% son correctas. A continuación, se representan los resultados en una tabla, donde se representa el instante de tiempo en el que se detecta el objeto y el identificador de cada objeto. El intervalo de tiempo entre cada detección va variando, realizándose aproximadamente cada 70 – 100 ms. Es el tiempo que tarda en procesar la nube de puntos: desde que el sensor la presencia de objetos, guarda los datos con la información RGB generando una nube de puntos RGB, segmenta los objetos y los filtra y calcula su descriptor VFH. Dependiendo de la cantidad de puntos que haya en cada escena, tardará más tiempo o menos.

Instante de tiempo (s)	Objeto detectado
0.207843	vfh_lidar_V2/peaton_175_vfh.pcd
0.300369	vfh_lidar_V2/coche_280_vfh.pcd
0.388660	vfh_lidar_V2/coche_290_vfh.pcd
0.466486	vfh_lidar_V2/coche_85_vfh.pcd
0.543738	vfh_lidar_V2/coche_100_vfh.pcd
0.623384	vfh_lidar_V2/coche_70_vfh.pcd
0.711273	vfh_lidar_V2/coche_80_vfh.pcd
0.797167	vfh_lidar_V2/coche_100_vfh.pcd
0.879418	vfh_lidar_V2/coche_100_vfh.pcd
0.965414	vfh_lidar_V2/coche_125_vfh.pcd
1.051436	vfh_lidar_V2/coche_280_vfh.pcd
1.151916	vfh_lidar_V2/coche_100_vfh.pcd
1.249937	vfh_lidar_V2/coche_105_vfh.pcd
1.344465	vfh_lidar_V2/coche_85_vfh.pcd
1.450463	vfh_lidar_V2/coche_105_vfh.pcd
1.548108	vfh_lidar_V2/coche_100_vfh.pcd
1.649777	vfh_lidar_V2/coche_95_vfh.pcd
1.744933	vfh_lidar_V2/coche_90_vfh.pcd
1.844765	vfh_lidar_V2/coche_95_vfh.pcd
1.943280	vfh_lidar_V2/peaton_135_vfh.pcd
2.044646	vfh_lidar_V2/coche_95_vfh.pcd
2.148151	vfh_lidar_V2/coche_290_vfh.pcd
2.244323	vfh_lidar_V2/coche_100_vfh.pcd
2.342273	vfh_lidar_V2/coche_125_vfh.pcd

2.438232	vfh_lidar_V2/coche_75_vfh.pcd
2.530382	vfh_lidar_V2/coche_260_vfh.pcd
2.627930	vfh_lidar_V2/coche_260_vfh.pcd
2.733881	vfh_lidar_V2/coche_265_vfh.pcd
2.820132	vfh_lidar_V2/coche_100_vfh.pcd
2.921485	vfh_lidar_V2/coche_280_vfh.pcd
3.012278	vfh_lidar_V2/peaton_135_vfh.pcd
3.107781	vfh_lidar_V2/coche_85_vfh.pcd
3.214816	vfh_lidar_V2/coche_280_vfh.pcd
3.315296	vfh_lidar_V2/coche_270_vfh.pcd
3.421247	vfh_lidar_V2/coche_100_vfh.pcd
3.521727	vfh_lidar_V2/coche_90_vfh.pcd
3.613295	vfh_lidar_V2/coche_85_vfh.pcd
3.709564	vfh_lidar_V2/coche_90_vfh.pcd
3.799623	vfh_lidar_V2/coche_285_vfh.pcd
3.874521	vfh_lidar_V2/coche_280_vfh.pcd
3.980472	vfh_lidar_V2/coche_270_vfh.pcd
4.080304	vfh_lidar_V2/coche_260_vfh.pcd
4.106840	vfh_lidar_V2/coche_280_vfh.pcd
4.203294	vfh_lidar_V2/coche_105_vfh.pcd
4.298423	vfh_lidar_V2/coche_270_vfh.pcd
4.386913	vfh_lidar_V2/peaton_135_vfh.pcd
4.480023	vfh_lidar_V2/coche_80_vfh.pcd
4.578630	vfh_lidar_V2/coche_105_vfh.pcd
4.669872	vfh_lidar_V2/coche_100_vfh.pcd
4.762803	vfh_lidar_V2/coche_95_vfh.pcd
4.840055	vfh_lidar_V2/coche_100_vfh.pcd
4.936578	vfh_lidar_V2/coche_110_vfh.pcd
5.016252	vfh_lidar_V2/peaton_175_vfh.pcd
5.123698	vfh_lidar_V2/coche_280_vfh.pcd
5.210682	vfh_lidar_V2/coche_100_vfh.pcd
5.311965	vfh_lidar_V2/coche_270_vfh.pcd
5.403672	vfh_lidar_V2/coche_270_vfh.pcd
5.487658	vfh_lidar_V2/peaton_175_vfh.pcd
5.578963	vfh_lidar_V2/coche_90_vfh.pcd
5.676201	vfh_lidar_V2/coche_275_vfh.pcd

Tabla 3: Resultados caso 1 con VLP

La orientación del objeto a detectar con la base de datos es de 270º, o en su defecto, la pose más similar sería en torno a los 90º. Se observan valores muy similares en torno a estos ángulos de giro.

5.3.1.2 Caso 1: Resultados con HDL64E

Se han tomado un total de 26 muestras, siendo el resultado correcto al 100%. El obstáculo es detectado siempre como un coche. A continuación, se representan los resultados en una tabla que contiene el instante de tiempo en el que se capturan los datos y el identificador del objeto detectado. En comparación con el caso del LIDAR, el tiempo de ejecución es mayor: el intervalo entre capturas es de aproximadamente 2 segundos. Este retardo es debido a que, además de los procesos que hay que realizar en los datos del LIDAR, se añade la conversión de la imagen de profundidad tomada por los sensores a la nube de puntos RGB. La nube de puntos resultante tiene mayor número de puntos debido a que los sensores son de mayor resolución, por lo que el tiempo de procesamiento de la nube es mayor.

Instante de tiempo (s)	Objeto detectado
0.188874	base datos camara R30/coche camera 20 vfh.pcd
2.114480	base_datos_camara_R30/coche_camera_30_vfh.pcd
4.098523	base_datos_camara_R30/coche_camera_20_vfh.pcd
6.040795	base_datos_camara_R30/coche_camera_30_vfh.pcd
7.893377	base datos camara R30/coche camera 30 vfh.pcd
9.939246	base_datos_camara_R30/coche_camera_330_vfh.pcd
11.951854	base_datos_camara_R30/coche_camera_30_vfh.pcd
13.927095	base_datos_camara_R30/coche_camera_150_vfh.pcd
15.831758	base datos camara R30/coche camera 30 vfh.pcd
17.825560	base_datos_camara_R30/coche_camera_20_vfh.pcd
19.622627	base datos camara R30/coche camera 20 vfh.pcd
21.660024	base_datos_camara_R30/coche_camera_20_vfh.pcd
23.522345	base datos camara R30/coche camera 20 vfh.pcd
25.325983	base_datos_camara_R30/coche_camera_20_vfh.pcd
27.279212	base_datos_camara_R30/coche_camera_20_vfh.pcd
29.185604	base_datos_camara_R30/coche_camera_150_vfh.pcd
31.071809	base datos camara R30/coche camera 20 vfh.pcd
32.886471	base_datos_camara_R30/coche_camera_20_vfh.pcd
34.808411	base_datos_camara_R30/coche_camera_20_vfh.pcd
36.628329	base_datos_camara_R30/coche_camera_20_vfh.pcd
38.473510	base datos camara R30/coche camera 150 vfh.pcd
40.341510	base_datos_camara_R30/coche_camera_150_vfh.pcd
42.345310	base datos camara R30/coche camera 30 vfh.pcd
44.200599	base_datos_camara_R30/coche_camera_20_vfh.pcd

46.101864	base datos camara R30/coche camera 20 vfh.pcd
47.951208	base_datos_camara_R30/coche_camera_330_vfh.pcd

Tabla 4: Resultados caso 1 con HDL

En cuanto a la posición, que debería ser detectada o a 90º o 270º, en muy pocos casos la detecta correctamente.

5.3.2 Caso 2: Vehículo en una intersección

El segundo caso de uso consistirá en realizar una simulación de un *ceda el paso*. El vehículo sensorizado se encuentra detenido a la entrada de la intersección mientras que un vehículo circula por el carril al cual el vehículo sensorizado quiere acceder.

Para ello, el vehículo rojo se mueve en dirección perpendicular al vehículo sensorizado, a una distancia de aproximadamente 8 metros, y con una velocidad de 5 m/s.

El movimiento queda definido como en la siguiente imagen, obtenida de la simulación:

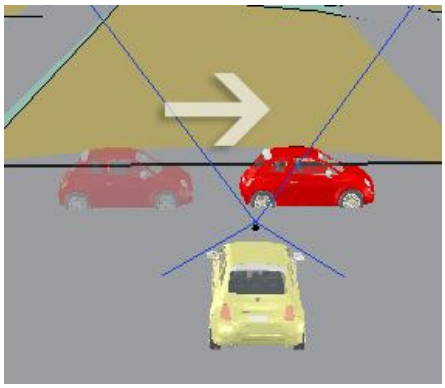


Ilustración 5-4: Caso 2 en V-REP

El rango de apertura de las cámaras es de 80º, por lo que no se podrá capturar toda la trayectoria del vehículo ya que está limitada por el rango de éstas.

Se sabe que el coche está separado por una distancia de 8 metros al vehículo sensorizado, y que los tres ángulos que forman el "triángulo" que es capaz de detectar el sensor deben sumar 180º, por lo que los otros dos ángulos serán de 50º cada uno.

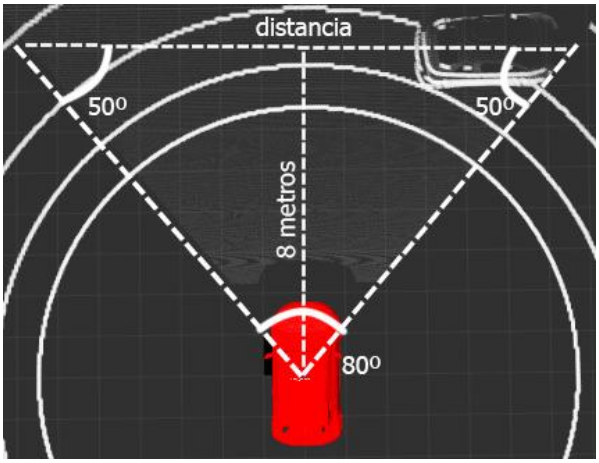


Ilustración 5-5: Cálculo de distancia al obstáculo

Empleando trigonometría, se calcula la distancia en horizontal en la que se podrá detectar a un vehículo situado a 8 metros:

Tg 50º = 8/(distancia/2)

Distancia = 13,43 metros.

Los sensores detectarán al vehículo circulando a lo largo de 13,43 metros aproximadamente. Si el vehículo circula a una velocidad de 5 m/s, la simulación durará:

Velocidad (m/s) = distancia (m) / tiempo (s)

Tiempo = distancia/velocidad=13,43/5 = 2,686 segundos

5.3.2.1 Caso 2: Resultados con VLP-16

Se almacenan los resultados obtenidos durante la simulación. En los primeros y últimos instantes de tiempo en los cuales el coche no es detectado completamente, se reconoce como un peatón. En el resto del tiempo, en el cual el vehículo aparece completo, se detecta como un vehículo. Durante esta simulación, se han obtenido 30 muestras de reconocimiento (desde que se comienza a detectar el objeto hasta que desaparece del rango de visión, siendo un 10% incorrectas y el 90% restante correctas.

Instante de tiempo (s)	Objeto detectado
0.0	vfh_lidar_V2/peaton_125_vfh.pcd
0.171764	vfh_lidar_V2/peaton_135_vfh.pcd
0.265166	vfh_lidar_V2/coche_100_vfh.pcd
0.342768	vfh_lidar_V2/coche_300_vfh.pcd
0.405524	vfh_lidar_V2/coche_290_vfh.pcd
0.473617	vfh_lidar_V2/coche_255_vfh.pcd
0.551315	vfh_lidar_V2/coche_245_vfh.pcd
0.626649	vfh_lidar_V2/coche_290_vfh.pcd
0.696621	vfh_lidar_V2/coche_290_vfh.pcd
0.776465	vfh_lidar_V2/coche_130_vfh.pcd
0.853646	vfh_lidar_V2/coche_350_vfh.pcd
0.932372	vfh_lidar_V2/coche_300_vfh.pcd
1.015283	vfh_lidar_V2/coche_185_vfh.pcd
1.102635	vfh_lidar_V2/coche_25_vfh.pcd
1.185920	vfh_lidar_V2/coche_300_vfh.pcd
1.267309	vfh_lidar_V2/coche_70_vfh.pcd
1.352604	vfh_lidar_V2/coche_165_vfh.pcd
1.433824	vfh_lidar_V2/coche_70_vfh.pcd
1.524189	vfh_lidar_V2/coche_165_vfh.pcd
1.610489	vfh_lidar_V2/coche_70_vfh.pcd
1.695221	vfh_lidar_V2/coche_125_vfh.pcd

1.782850	vfh_lidar_V2/coche_205_vfh.pcd
1.860390	vfh_lidar_V2/coche_40_vfh.pcd
1.952142	vfh_lidar_V2/coche_195_vfh.pcd
2.038391	vfh_lidar_V2/coche_115_vfh.pcd
2.109097	vfh_lidar_V2/coche_150_vfh.pcd
2.197125	vfh_lidar_V2/coche_40_vfh.pcd
2.268618	vfh_lidar_V2/coche_110_vfh.pcd
2.356083	vfh_lidar_V2/coche_90_vfh.pcd
2.432322	vfh_lidar_V2/peaton_175_vfh.pcd

Tabla 5: Resultados caso 2 con VLP

En cuanto a la posición del objeto, en referencia con la base de datos se encuentra a 0º (o muy similar, a 180º), pero el obstáculo se va moviendo, y por tanto la perspectiva va variando.

5.3.2.2 Caso 2: Resultados con HDL64E

Se obtienen un total de 28 resultados, donde los 27 primeros son detectados como vehículo y en el último caso como un peatón. Esto es debido a que en el último instante de la simulación el vehículo no es capturado completamente, sino solo una parte de él y por tanto el número de puntos es menor. La probabilidad de acierto es de 96,43% frente a la de fallo de un 3,57%.

Instante de tiempo (s)	Objeto detectado
0.133912	base_datos_camara_R30/coche_camera_220_vfh.pcd
1.838838	base_datos_camara_R30/coche_camera_40_vfh.pcd
3.456321	base datos camara R30/coche camera 40 vfh.pcd
5.102874	base_datos_camara_R30/coche_camera_40_vfh.pcd
6.716229	base_datos_camara_R30/coche_camera_40_vfh.pcd
8.313843	base_datos_camara_R30/coche_camera_330_vfh.pcd
9.899601	base datos camara R30/coche camera 30 vfh.pcd
11.503667	base_datos_camara_R30/coche_camera_40_vfh.pcd
13.047959	base_datos_camara_R30/coche_camera_40_vfh.pcd
14.623262	base_datos_camara_R30/coche_camera_20_vfh.pcd
16.376104	base datos camara R30/coche camera 20 vfh.pcd
18.069997	base_datos_camara_R30/coche_camera_350_vfh.pcd
19.653224	base_datos_camara_R30/coche_camera_350_vfh.pcd
21.339075	base_datos_camara_R30/coche_camera_350_vfh.pcd
22.916981	base datos camara R30/coche camera 350 vfh.pcd
24.485650	base_datos_camara_R30/coche_camera_20_vfh.pcd
26.147731	base_datos_camara_R30/coche_camera_20_vfh.pcd
27.892457	base_datos_camara_R30/coche_camera_200_vfh.pcd
29.567084	base datos camara R30/coche camera 20 vfh.pcd

31.322108	base_datos_camara_R30/coche_camera_30_vfh.pcd
32.998778	base_datos_camara_R30/coche_camera_30_vfh.pcd
34.649195	base_datos_camara_R30/coche_camera_200_vfh.pcd
36.398806	base datos camara R30/coche camera 330 vfh.pcd
38.010545	base_datos_camara_R30/coche_camera_40_vfh.pcd
39.550124	base_datos_camara_R30/coche_camera_210_vfh.pcd
41.164716	base_datos_camara_R30/coche_camera_330_vfh.pcd
42.954284	base datos camara R30/coche camera 50 vfh.pcd
44.648233	base_datos_camara_R30/peaton_camera_20_vfh.pcd

Tabla 6: Resultados caso 2 con HDL

El ángulo a detectar sería de 0º o 180º, en este caso detecta valores más similares (20, 30, 200).

5.3.3 Caso 3: Vehículo en sentido contrario

El tercer caso de reconocimiento de vehículos será circular por una vía de doble sentido.

Para ello, se hace circular un vehículo en sentido contrario por el carril izquierdo en posición al vehículo sensorizado. El vehículo a detectar circula a una velocidad de 10 m/s, mientras que el vehículo sensorizado circula a 5 m/s.

Se crea una trayectoria en línea recta para el vehículo que se quiere detectar, donde inicialmente se encuentra a aproximadamente 28 metros. El vehículo avanza hasta pasar por el lado del vehículo sensorizado.

La escena resultante en V-REP queda de la siguiente forma:

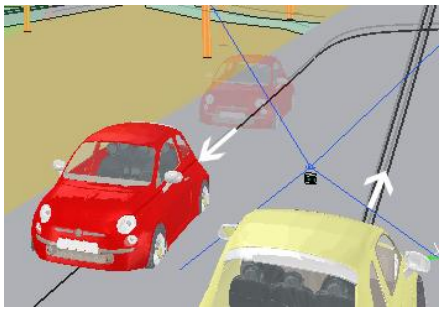


Ilustración 5-6: Caso 3 en V-REP

5.3.3.1 Caso 3: Resultados con VLP-16

En este caso es en el que menos datos de reconocimiento se obtienen, ya que al avanzar los dos vehículos con diferentes velocidades y en sentido contrario, se pierde a los pocos segundos. Se han obtenido solo 13 resultados, siendo el 38,46% erróneos frente al 61,54% correctos. Esto es debido a que durante los primeros instantes de la simulación detecta pocos puntos del objeto debido a la lejanía.

Instante de tiempo (s)	Objeto detectado
0.130024	vfh_lidar_V2/peaton_125_vfh.pcd
0.216830	vfh_lidar_V2/peaton_125_vfh.pcd
0.305095	vfh_lidar_V2/coche_80_vfh.pcd
0.377578	vfh_lidar_V2/coche_180_vfh.pcd
0.455422	vfh_lidar_V2/coche_70_vfh.pcd
0.530403	vfh_lidar_V2/peaton_125_vfh.pcd
0.596007	vfh_lidar_V2/coche_260_vfh.pcd
0.669384	vfh_lidar_V2/coche_115_vfh.pcd
0.749809	vfh_lidar_V2/coche_180_vfh.pcd
0.836172	vfh_lidar_V2/coche_115_vfh.pcd
0.922919	vfh_lidar_V2/peaton_135_vfh.pcd
1.011049	vfh_lidar_V2/coche_85_vfh.pcd
1.088010	vfh_lidar_V2/peaton_125_vfh.pcd

Tabla 7: Resultados caso 3 con VLP

La orientación del objeto en referencia a la base de datos es de 270º (o de forma similar, 90º). Salvo en 2 casos que lo detecta con una pose de 180º, el resto está en el rango marcado ±25º.

5.3.3.2 Caso 3: Resultados con HDL64E

Se han obtenido un total de 22 resultados, siendo todos correctos. A diferencia del caso anterior, al tener más puntos de cada objeto, se detectan más muestras de él y con más claridad, con un 100% de probabilidad de acierto.

Instante de tiempo (s)	Objeto detectado
0.153282	base_datos_camara_R30/coche_camera_190_vfh.pcd
1.994267	base_datos_camara_R30/coche_camera_190_vfh.pcd
3.877844	base_datos_camara_R30/coche_camera_270_vfh.pcd
5.820066	base_datos_camara_R30/coche_camera_80_vfh.pcd
7.831034	base_datos_camara_R30/coche_camera_330_vfh.pcd
9.836574	base_datos_camara_R30/coche_camera_190_vfh.pcd
11.821797	base_datos_camara_R30/coche_camera_110_vfh.pcd
13.830568	base_datos_camara_R30/coche_camera_270_vfh.pcd
15.627969	base_datos_camara_R30/coche_camera_190_vfh.pcd
17.628758	base_datos_camara_R30/coche_camera_80_vfh.pcd
19.623372	base_datos_camara_R30/coche_camera_270_vfh.pcd
21.690518	base_datos_camara_R30/coche_camera_210_vfh.pcd
23.695571	base_datos_camara_R30/coche_camera_300_vfh.pcd
25.637726	base_datos_camara_R30/coche_camera_300_vfh.pcd

25.641547	base_datos_camara_R30/coche_camera_220_vfh.pcd
27.664681	base_datos_camara_R30/coche_camera_210_vfh.pcd
29.638761	base_datos_camara_R30/coche_camera_310_vfh.pcd
31.512439	base_datos_camara_R30/coche_camera_130_vfh.pcd
33.480825	base_datos_camara_R30/coche_camera_230_vfh.pcd
35.299160	base_datos_camara_R30/coche_camera_300_vfh.pcd
37.203649	base_datos_camara_R30/coche_camera_300_vfh.pcd
39.224413	base_datos_camara_R30/coche_camera_270_vfh.pcd

Tabla 8: Resultados caso 3 con HDL

El valor del ángulo debería ser de 90º o 270º. En algunos casos lo detecta correctamente, pero se encuentran algunos casos donde el ángulo detectado es de 190º.

5.3.4 Caso 4: Paso de peatones

El primer caso que se analizará con peatones será el de paso de peatón: el vehículo sensorizado se encuentra detenido, mientras el peatón cruza la carretera. El peatón pasará a una distancia de 5,4 metros por delante del vehículo sensorizado, con una velocidad de 0,56 m/s, realizando una trayectoria en línea recta.

El filtrado por color del peatón se realiza segmentando el color de la camiseta, por lo que para este caso y los posteriores, se pintará el peatón entero del mismo color para tener más precisión en los resultados.

La escena en V-REP queda de la siguiente forma:

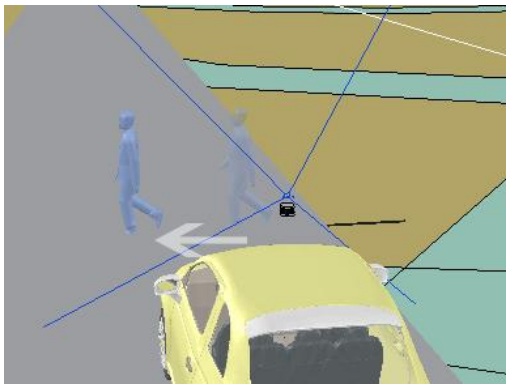


Ilustración 5-7: Caso 4 en V-REP.

Siguiendo el mismo planteamiento del caso 2, se calcula la distancia recorrida por el peatón que podrá ser capturada por el sensor. Se encuentra separado del vehículo por 5,4 metros, por tanto la distancia será:

Tg 50º = 5,4/(distancia/2)

Distancia = 9,06 metros.

Puesto que la velocidad del peatón es de 0,56 m/s, el tiempo de la simulación deberá ser de:

Velocidad (m/s) = distancia (m) / tiempo (s)

Tiempo = distancia/velocidad=9,06/0,56 = 16,18 segundos

5.3.4.1 Caso 4: Resultados con VLP-16

Se almacenan los resultados de la simulación. En este caso, se han almacenado 64 datos de reconocimiento en los que es visible el peatón. En 3 de estas ocasiones, se detecta al peatón como un vehículo, mientras que en las 61 restantes es detectado como un peatón con un ángulo de giro de 125º (salvo en una que lo detecta con un ángulo de giro de 175º). Se obtiene una probabilidad de acierto del 95,31% frente al 4,69% de fallo.

Instante de tiempo	Objeto detectado
0.145101	vfh_lidar_V2/peaton_125_vfh.pcd
0.240576	vfh_lidar_V2/peaton_125_vfh.pcd
0.305739	vfh_lidar_V2/peaton_125_vfh.pcd
0.368911	vfh_lidar_V2/peaton_125_vfh.pcd
0.427623	vfh_lidar_V2/peaton_125_vfh.pcd
0.499512	vfh_lidar_V2/peaton_125_vfh.pcd
0.563418	vfh_lidar_V2/peaton_125_vfh.pcd
0.630235	vfh_lidar_V2/peaton_125_vfh.pcd
0.696528	vfh_lidar_V2/peaton_125_vfh.pcd
0.770080	vfh_lidar_V2/peaton_125_vfh.pcd
0.837903	vfh_lidar_V2/peaton_125_vfh.pcd
0.910981	vfh_lidar_V2/peaton_125_vfh.pcd
0.987660	vfh_lidar_V2/peaton_125_vfh.pcd
1.058859	vfh_lidar_V2/peaton_125_vfh.pcd
1.124123	vfh_lidar_V2/peaton_125_vfh.pcd
1.193352	vfh_lidar_V2/peaton_125_vfh.pcd
1.273188	vfh_lidar_V2/peaton_125_vfh.pcd
1.353403	vfh_lidar_V2/coche_95_vfh.pcd
1.420280	vfh_lidar_V2/peaton_125_vfh.pcd
1.495280	vfh_lidar_V2/peaton_125_vfh.pcd
1.570891	vfh_lidar_V2/peaton_125_vfh.pcd
1.647564	vfh_lidar_V2/peaton_125_vfh.pcd
1.726043	vfh_lidar_V2/peaton_125_vfh.pcd
1.793536	vfh_lidar_V2/peaton_125_vfh.pcd
1.867070	vfh_lidar_V2/peaton_125_vfh.pcd
1.931856	vfh_lidar_V2/peaton_125_vfh.pcd
2.009234	vfh_lidar_V2/peaton_125_vfh.pcd
2.081372	vfh_lidar_V2/peaton_125_vfh.pcd

2.154313	vfh_lidar_V2/peaton_125_vfh.pcd
2.232196	vfh_lidar_V2/peaton_125_vfh.pcd
2.307368	vfh_lidar_V2/peaton_125_vfh.pcd
2.383308	vfh_lidar_V2/peaton_125_vfh.pcd
2.452009	vfh_lidar_V2/coche_90_vfh.pcd
2.519166	vfh_lidar_V2/peaton_125_vfh.pcd
2.589716	vfh_lidar_V2/peaton_125_vfh.pcd
2.655787	vfh_lidar_V2/peaton_125_vfh.pcd
2.729165	vfh_lidar_V2/peaton_125_vfh.pcd
2.802749	vfh_lidar_V2/peaton_125_vfh.pcd
2.871591	vfh_lidar_V2/peaton_125_vfh.pcd
2.936384	vfh_lidar_V2/peaton_125_vfh.pcd
3.015798	vfh_lidar_V2/peaton_125_vfh.pcd
3.086705	vfh_lidar_V2/peaton_125_vfh.pcd
3.157327	vfh_lidar_V2/peaton_125_vfh.pcd
3.230574	vfh_lidar_V2/coche_95_vfh.pcd
3.300366	vfh_lidar_V2/peaton_125_vfh.pcd
3.379233	vfh_lidar_V2/peaton_125_vfh.pcd
3.449382	vfh_lidar_V2/peaton_125_vfh.pcd
3.517359	vfh_lidar_V2/peaton_125_vfh.pcd
3.589855	vfh_lidar_V2/peaton_125_vfh.pcd
3.651271	vfh_lidar_V2/peaton_125_vfh.pcd
3.719632	vfh_lidar_V2/peaton_125_vfh.pcd
3.793006	vfh_lidar_V2/peaton_125_vfh.pcd
3.864458	vfh_lidar_V2/peaton_125_vfh.pcd
3.936008	vfh_lidar_V2/peaton_125_vfh.pcd
4.006190	vfh_lidar_V2/peaton_175_vfh.pcd
4.091642	vfh_lidar_V2/peaton_125_vfh.pcd
4.163715	vfh_lidar_V2/peaton_125_vfh.pcd
4.239371	vfh_lidar_V2/peaton_125_vfh.pcd
4.314053	vfh_lidar_V2/peaton_125_vfh.pcd
4.380874	vfh_lidar_V2/peaton_125_vfh.pcd
4.461651	vfh_lidar_V2/peaton_125_vfh.pcd
4.526789	vfh_lidar_V2/peaton_125_vfh.pcd
4.607768	vfh_lidar_V2/peaton_125_vfh.pcd
4.676059	vfh_lidar_V2/peaton_125_vfh.pcd

Tabla 9: Resultados del caso 4 en VLP

En este caso, obtener la orientación es más complicado que en los casos anteriores, ya que la nube de puntos del peatón contiene menos puntos y por tanto menos información. Se detecta siempre con la misma orientación (125º). El giro debería ser de 90º, y se detecta con un error de 35º.

5.3.4.2 Caso 4: Resultados con HDL64E

Se han obtenido 54 resultados, siendo 10 de ellos erróneos (detectando al peatón como un vehículo), y 44 correctos. En este caso, la probabilidad de acierto ha sido de un 81,5% y la de fallo un 18,5%.

Instante de tiempo (s)	Objeto detectado
0.117690	base datos camara R30/coche camera 20 vfh.pcd
1.735841	base_datos_camara_R30/peaton_camera_10_vfh.pcd
3.415063	base datos camara R30/peaton camera 0 vfh.pcd
5.138886	base_datos_camara_R30/coche_camera_20_vfh.pcd
6.862412	base datos camara R30/peaton camera 110 vfh.pcd
8.591039	base_datos_camara_R30/coche_camera_350_vfh.pcd
10.271818	base_datos_camara_R30/coche_camera_20_vfh.pcd
11.923596	base_datos_camara_R30/peaton_camera_240_vfh.pcd
13.623662	base_datos_camara_R30/peaton_camera_110_vfh.pcd
15.357511	base_datos_camara_R30/coche_camera_230_vfh.pcd
17.175078	base datos camara R30/coche camera 230 vfh.pcd
18.896087	base_datos_camara_R30/peaton_camera_50_vfh.pcd
20.642592	base_datos_camara_R30/peaton_camera_60_vfh.pcd
22.325800	base_datos_camara_R30/peaton_camera_140_vfh.pcd
24.112809	base datos camara R30/peaton camera 290 vfh.pcd
25.845951	base_datos_camara_R30/peaton_camera_140_vfh.pcd
27.613800	base_datos_camara_R30/peaton_camera_150_vfh.pcd
29.399651	base_datos_camara_R30/peaton_camera_300_vfh.pcd
31.208295	base_datos_camara_R30/peaton_camera_250_vfh.pcd
32.960115	base_datos_camara_R30/coche_camera_230_vfh.pcd
34.577023	base_datos_camara_R30/peaton_camera_240_vfh.pcd
36.305167	base_datos_camara_R30/peaton_camera_140_vfh.pcd
38.016936	base datos camara R30/peaton camera 80 vfh.pcd
39.881151	base_datos_camara_R30/peaton_camera_80_vfh.pcd
41.763591	base_datos_camara_R30/coche_camera_110_vfh.pcd
43.488251	base_datos_camara_R30/peaton_camera_50_vfh.pcd
45.173217	base datos camara R30/peaton camera 230 vfh.pcd

46.941651	base_datos_camara_R30/peaton_camera_50_vfh.pcd
48.592491	base_datos_camara_R30/peaton_camera_240_vfh.pcd
50.324995	base_datos_camara_R30/peaton_camera_60_vfh.pcd
52.079174	base datos camara R30/coche camera 110 vfh.pcd
53.810603	base_datos_camara_R30/peaton_camera_230_vfh.pcd
55.569956	base_datos_camara_R30/peaton_camera_290_vfh.pcd
57.328923	base_datos_camara_R30/peaton_camera_50_vfh.pcd
59.146035	base datos camara R30/peaton camera 290 vfh.pcd
60.694975	base_datos_camara_R30/peaton_camera_50_vfh.pcd
62.406961	base_datos_camara_R30/peaton_camera_230_vfh.pcd
64.209500	base_datos_camara_R30/peaton_camera_20_vfh.pcd
65.903204	base datos camara R30/coche camera 230 vfh.pcd
67.768282	base_datos_camara_R30/peaton_camera_300_vfh.pcd
69.399549	base_datos_camara_R30/peaton_camera_60_vfh.pcd
71.025443	base_datos_camara_R30/peaton_camera_100_vfh.pcd
72.664587	base datos camara R30/peaton camera 230 vfh.pcd
74.341136	base_datos_camara_R30/peaton_camera_240_vfh.pcd
76.048170	base_datos_camara_R30/peaton_camera_20_vfh.pcd
77.735246	base_datos_camara_R30/peaton_camera_20_vfh.pcd
79.499177	base datos camara R30/peaton camera 230 vfh.pcd
81.107690	base_datos_camara_R30/peaton_camera_230_vfh.pcd
82.837297	base_datos_camara_R30/peaton_camera_290_vfh.pcd
84.446130	base_datos_camara_R30/peaton_camera_240_vfh.pcd
86.123658	base datos camara R30/peaton camera 110 vfh.pcd
87.857014	base_datos_camara_R30/peaton_camera_110_vfh.pcd
89.549572	base_datos_camara_R30/peaton_camera_240_vfh.pcd
91.138801	base_datos_camara_R30/peaton_camera_240_vfh.pcd

Tabla 10: Resultados del caso 4 con VLP

5.3.5 Caso 5: Peatón por arcén derecho.

Para este caso, se escoge la escena del caso 3, donde se encuentra un vehículo circulando en sentido contrario a 10 m/s por el carril izquierdo, mientras que el vehículo sensorizado circula a 5 m/s por el carril derecho. Se añade un peatón caminando en sentido contrario que el vehículo sensorizado, por el arcén derecho.

La escena en V-REP queda de la siguiente forma:

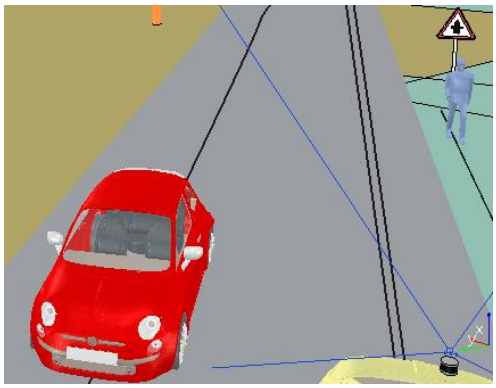


Ilustración 5-8: Caso 5 en V-REP.

5.3.5.1 Caso 5: Resultados con VLP-16

Se almacenan los resultados de la detección del peatón de este caso, ya que el reconocimiento del vehículo se estudió en los casos anteriores. Se han tomado 25 muestras de resultados, de las cuales únicamente en una ocasión se detecta al peatón como un vehículo, siendo la probabilidad de fallo en este caso del 4% frente al 96% de probabilidad de acierto.

Instante de tiempo (s)	Objeto detectado
0.0	vfh_lidar_V2/peaton_125_vfh.pcd
0.124237	vfh_lidar_V2/peaton_125_vfh.pcd
0.235927	vfh_lidar_V2/peaton_125_vfh.pcd
0.504009	vfh_lidar_V2/peaton_125_vfh.pcd
0.583010	vfh_lidar_V2/peaton_125_vfh.pcd
0.775854	vfh_lidar_V2/peaton_125_vfh.pcd
0.868159	vfh_lidar_V2/peaton_125_vfh.pcd
0.974749	vfh_lidar_V2/peaton_125_vfh.pcd
1.036868	vfh_lidar_V2/coche_35_vfh.pcd
1.119531	vfh_lidar_V2/peaton_125_vfh.pcd
1.123300	vfh_lidar_V2/peaton_125_vfh.pcd
1.219677	vfh_lidar_V2/peaton_125_vfh.pcd
1.308538	vfh_lidar_V2/peaton_125_vfh.pcd
1.474749	vfh_lidar_V2/peaton_125_vfh.pcd
1.559469	vfh_lidar_V2/peaton_125_vfh.pcd
1.563066	vfh_lidar_V2/peaton_125_vfh.pcd

1.659438	vfh_lidar_V2/peaton_175_vfh.pcd
1.746694	vfh_lidar_V2/peaton_125_vfh.pcd
1.846137	vfh_lidar_V2/peaton_125_vfh.pcd
1.942195	vfh_lidar_V2/peaton_125_vfh.pcd
2.028498	vfh_lidar_V2/peaton_125_vfh.pcd
2.112350	vfh_lidar_V2/peaton_125_vfh.pcd
2.195672	vfh_lidar_V2/peaton_125_vfh.pcd
2.298102	vfh_lidar_V2/peaton_125_vfh.pcd
2.396841	vfh_lidar_V2/peaton_125_vfh.pcd

Tabla 11: Resultados caso 4 con HDL

Al igual que en el caso 4, los puntos de la muestra obtenidos son muy pocos. Debería detectarse con un giro de 180º, pero se encuentra un error de 55º ya que se detecta siempre con el mismo giro.

5.3.5.2 Caso 5: Resultados con HDL64E

Se almacenan los resultados de la detección del peatón al igual que en el apartado anterior. Se han obtenido un total de 37 muestras, de las cuales 28 veces se detecta como un peatón, y 9 veces como un vehículo. La probabilidad de fallo ha aumentado, siendo de un 24,32% frente al caso anterior, y la de acierto de un 75,68%.

Instante de tiempo (s)	Objeto detectado
0.132312	base datos camara R30/peaton camera 140 vfh .pcd
1.979131	base_datos_camara_R30/coche_camera_190_vfh. pcd
3.894456	base_datos_camara_R30/coche_camera_180_vfh. pcd
5.917382	base_datos_camara_R30/coche_camera_80_vfh.p cd
7.749862	base datos camara R30/peaton camera 10 vfh. pcd
9.556718	base_datos_camara_R30/peaton_camera_160_vfh .pcd
11.560926	base_datos_camara_R30/peaton_camera_160_vfh .pcd
13.490405	base_datos_camara_R30/peaton_camera_160_vfh .pcd
15.456318	base datos camara R30/coche camera 270 vfh. pcd
17.269963	base_datos_camara_R30/peaton_camera_290_vfh .pcd
19.140298	base_datos_camara_R30/peaton_camera_50_vfh. pcd
21.157250	base_datos_camara_R30/peaton_camera_140_vfh .pcd
23.084654	base datos camara R30/peaton camera 140 vfh .pcd
25.109792	base_datos_camara_R30/peaton_camera_280_vfh .pcd
27.092605	base_datos_camara_R30/peaton_camera_290_vfh .pcd
29.152303	base_datos_camara_R30/peaton_camera_170_vfh .pcd
31.193701	base datos camara R30/peaton camera 160 vfh .pcd
33.272397	base_datos_camara_R30/coche_camera_180_vfh. pcd

35.200541	base datos camara R30/peaton camera 160 vfh .pcd
37.179685	base_datos_camara_R30/coche_camera_300_vfh. pcd
39.168322	base_datos_camara_R30/peaton_camera_150_vfh .pcd
41.037478	base_datos_camara_R30/peaton_camera_170_vfh .pcd
42.924110	base_datos_camara_R30/peaton_camera_220_vfh .pcd
45.005357	base_datos_camara_R30/peaton_camera_300_vfh .pcd
47.041752	base_datos_camara_R30/peaton_camera_300_vfh .pcd
48.910893	base_datos_camara_R30/peaton_camera_290_vfh .pcd
52.641247	base datos camara R30/coche camera 270 vfh. pcd
52.645939	base_datos_camara_R30/coche_camera_350_vfh. pcd
54.535043	base_datos_camara_R30/coche_camera_190_vfh. pcd
54.540234	base_datos_camara_R30/peaton_camera_10_vfh. pcd
56.413125	base datos camara R30/peaton camera 0 vfh.p cd
58.390247	base_datos_camara_R30/peaton_camera_160_vfh .pcd
60.453613	base_datos_camara_R30/peaton_camera_160_vfh .pcd
62.387371	base_datos_camara_R30/peaton_camera_340_vfh .pcd
64.435506	base datos camara R30/peaton camera 200 vfh .pcd
66.304830	base_datos_camara_R30/peaton_camera_340_vfh .pcd
68.241837	base_datos_camara_R30/peaton_camera_160_vfh .pcd

Tabla 12: Resultados del caso 5 con VLP

5.3.6 Caso 6: Peatón por arcén izquierdo.

Para este caso, se añadirá al caso 1 (vehículo circulando en el mismo sentido de circulación que el vehículo sensorizado), un peatón caminando por el arcén izquierdo con el mismo sentido. Los dos vehículos llevan la misma velocidad (5 m/s) y circulan por el carril derecho. El peatón camina por el arcén izquierdo, a una velocidad de 0,56 m/s siguiendo una trayectoria recta.

La escena resultante queda de la siguiente forma en el simulador V-REP:

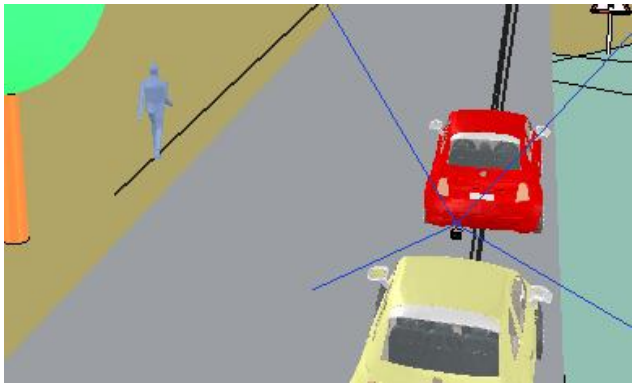


Ilustración 5-9: Caso 6 en V-REP.

5.3.6.1 Caso 6: Resultados con VLP-16

Se almacenan los resultados del reconocimiento del peatón, con un total de 18 resultados correctos. En este caso se ha obtenido un 100% de probabilidad de acierto, detectando al peatón en todos los casos con un giro de 125º, cuando debería de ser con un giro de 0º

Instante de tiempo (s)	Objeto detectado
0.127452	vfh_lidar_V2/peaton_125_vfh.pcd
0.200735	vfh_lidar_V2/peaton_125_vfh.pcd
0.356734	vfh_lidar_V2/peaton_125_vfh.pcd
0.439524	vfh_lidar_V2/peaton_125_vfh.pcd
0.540035	vfh_lidar_V2/peaton_125_vfh.pcd
0.683112	vfh_lidar_V2/peaton_125_vfh.pcd
0.825612	vfh_lidar_V2/peaton_125_vfh.pcd
0.925030	vfh_lidar_V2/peaton_125_vfh.pcd
1.139170	vfh_lidar_V2/peaton_125_vfh.pcd
1.235570	vfh_lidar_V2/peaton_125_vfh.pcd
1.393835	vfh_lidar_V2/peaton_125_vfh.pcd
1.538506	vfh_lidar_V2/peaton_125_vfh.pcd
1.633299	vfh_lidar_V2/peaton_125_vfh.pcd
1.724826	vfh_lidar_V2/peaton_125_vfh.pcd
1.828057	vfh_lidar_V2/peaton_125_vfh.pcd
2.025180	vfh_lidar_V2/peaton_125_vfh.pcd
2.122876	vfh_lidar_V2/peaton_125_vfh.pcd
2.227788	vfh_lidar_V2/peaton_125_vfh.pcd

Tabla 13: Resultados del caso 6 con VLP

5.3.6.2 Caso 6: Resultados con HDL64E

Se almacenan los resultados de la detección del peatón con la cámara para este caso, obteniéndose un total de 18 resultados. Todos los resultados son obtenidos correctamente, por lo que detecta al objeto como peatón.

Instante de tiempo (s)	Objeto detectado
0.140447	base_datos_camara_R30/peaton_camera_140_vfh.pcd
2.045194	base_datos_camara_R30/peaton_camera_250_vfh.pcd
4.086520	base_datos_camara_R30/peaton_camera_340_vfh.pcd
6.263545	base_datos_camara_R30/peaton_camera_140_vfh.pcd
10.556137	base_datos_camara_R30/peaton_camera_10_vfh.pcd
12.485057	base_datos_camara_R30/peaton_camera_140_vfh.pcd
14.460217	base_datos_camara_R30/peaton_camera_10_vfh.pcd

22.427573	base_datos_camara_R30/peaton_camera_10_vfh.pcd
24.424160	base_datos_camara_R30/peaton_camera_340_vfh.pcd
26.442520	base_datos_camara_R30/peaton_camera_10_vfh.pcd
28.489906	base_datos_camara_R30/peaton_camera_10_vfh.pcd
30.542703	base_datos_camara_R30/peaton_camera_10_vfh.pcd
32.549958	base_datos_camara_R30/peaton_camera_330_vfh.pcd
36.352071	base_datos_camara_R30/peaton_camera_230_vfh.pcd
38.465873	base_datos_camara_R30/peaton_camera_140_vfh.pcd
40.524104	base_datos_camara_R30/peaton_camera_110_vfh.pcd
46.502468	base_datos_camara_R30/peaton_camera_110_vfh.pcd
50.516356	base_datos_camara_R30/peaton_camera_0_vfh.pcd

Tabla 14: Resultados del caso 6 con HDL

6 CONCLUSIONES

En este trabajo de fin de grado, se ha implementado un algoritmo capaz de extraer los objetos de una escena que rodean a un vehículo mediante diferentes sensores, e identificarlos mediante los descriptores VFH.

Se han llegado a las siguientes conclusiones tras finalizar este proyecto:

Mediante la creación de una base de datos de los diferentes objetos presentes en el entorno, se pueden reconocer objetos mediante la comparación (descriptores VFH). Este reconocimiento es más preciso si los objetos no se encuentran en movimiento. Si los diferentes obstáculos se encuentran en movimiento, la nube de datos se puede distorsionar y perder información, de modo que el reconocimiento a veces falle.

Se han creado 6 casos en el simulador VREP, en los que se encuentran vehículos y peatones en movimiento, y se realiza el reconocimiento de los objetos presentes en las escenas desde el momento que comienzan a ser capturados por los sensores hasta que dejan de ser visibles. Durante algunos tramos de la secuencia de la simulación, se detecta incorrectamente el tipo de objeto, debido a que la nube de puntos que se analiza (combinación de sensor + color) en ocasiones sale distorsionada (en la nube de puntos de color a veces se obtiene una sombra). Es por ello que en los datos obtenidos por las cámaras, se selecciona el modelo con radio de búsqueda de 30 cm y sin escalado de la nube: para eliminar la sombra se filtra la nube de puntos total en altura, perdiéndose parte de la información de los objetos, por lo que el reconocimiento falla. Se realizaron pruebas con la base de datos escalada con un radio de búsqueda de 30 cm: con filtrado en altura de 1,3 metros para eliminar la sombra, los vehículos se detectan en varias ocasiones como peatones, mientras que si el filtrado en altura es de 1,5 metros, la detección es correcta pero en los casos en los que los obstáculos se acercan al sensor, la sombres detectada como otro objeto. Es por esto, que se obtuvieron mejores resultados con un filtrado en altura de 1,3 metros, radio de búsqueda de 30 cm sin escalar.

A continuación, se muestran algunos ejemplos en los que la detección es correcta y otros en los que es incorrecta (de los casos creados). Para ello se emplea el visualizador RViz, en los cuales se muestra un topic de un *marker* que indica la posición donde es detectado un objeto, y otro de un texto identificativo que muestra qué objeto es.

En el caso de que no aparezca ninguna sombra, el reconocimiento y seguimiento del objeto se realiza correctamente, como se observa en la siguiente imagen durante la simulación del caso 2:

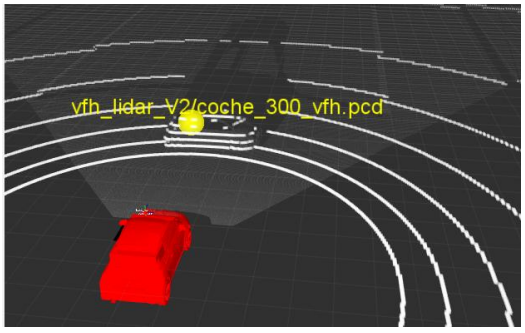


Ilustración 6-1: Visualización en RViz caso 2

En los casos en los que el obstáculo se acerca, se proyecta una sombra debido a los errores de calibración de la cámara, de forma que se detecta como otro obstáculo.

En el caso 3, el obstáculo es un vehículo que se acerca. En ocasiones se proyecta una sombra que es detectada como otro objeto, lo que hace que el objeto original se deforme y sea reconocido de forma incorrecta, como ocurre en la siguiente imagen:

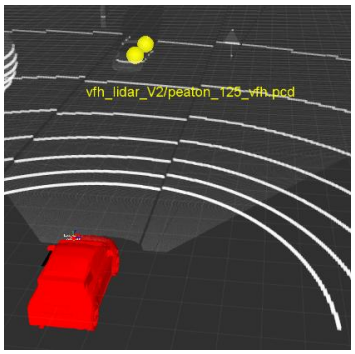


Ilustración 6-2: Visualización en RViz del caso 3

Por la proyección, el obstáculo se ha detectado como 2 objetos (aparecen 2 puntos en lugar de 1), y es identificado como un peatón (debido a que el objeto aparece deformado).

En el caso 5, en el que aparece un peatón circulando por el arcén derecho (acercándose al vehículo sensorizado), también se proyecta una sombra en múltiples ocasiones. Como se puede observar en la primera imagen correspondiente al inicio de la simulación, el obstáculo es identificado correctamente como un peatón, y por tanto, el seguimiento también es correcto.

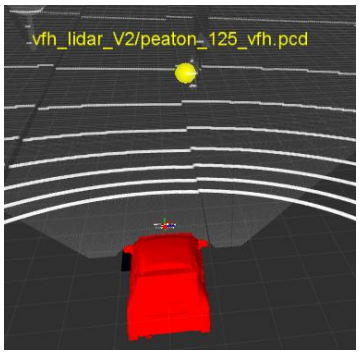


Ilustración 6-3: Visualización en RViz del caso 5 correcta

Al avanzar la simulación, se observa que se proyectan diversas sombras del peatón, lo que hace que se detecten más obstáculos de los que hay, por lo que la identificación y el seguimiento fallan, como se observa en la siguiente imagen:

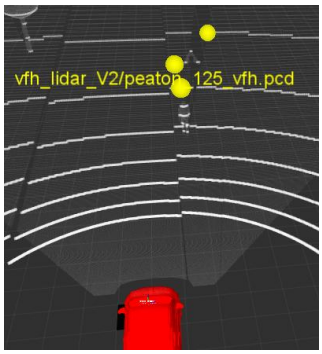


Ilustración 6-4: Visualización en RViz del caso 5 con fallos

Además, se aprecia que en las pruebas sin color donde se detectaban objetos detenidos, la detección era correcta debido a que no existe la sombra. A continuación se exponen los resultados en una tabla, donde los 3 primeros casos se detecta un vehículo y en los siguientes un peatón:

	VLP 16		HDL 64E	
	Acierto (%)	Fallo (%)	Acierto (%)	Fallo (%)
Caso 1	90	10	100	0
Caso 2	90	10	96,43	3,57
Caso 3	61,54	38,46	100	0
Caso 4	95,31	4,69	81,5	18,5
Caso 5	96	4	75,68	24,32
Caso 6	100	0	100	0

Tabla 15: Comparativa de los resultados

Se observa que la detección de vehículos es más acertada con el uso de la cámara, corrigiéndose en el tercer caso de un 61,54% de acierto a un 100%. Sin embargo, la detección del peatón es más acertada con el uso del LIDAR, aunque los porcentajes de acierto con la cámara son muy altos.

7 MANUAL DE USUARIO

Se explicarán los pasos a seguir para lanzar la aplicación: desde la creación de la base de datos hasta el reconocimiento de objetos en distintos casos.

Para utilizar el sistema ROS, en la terminal se deberá emplear el siguiente comando que inicializa el Master, los parámetros de servicio, y al sistema de logging de nodos que permitirá suscribirse y publicar

```
~ roscore
```

7.1 BASE DE DATOS

Para crear la base de datos, en primer lugar se almacenarán los archivos .pcd que contienen las nubes de puntos de los objetos que formarán parte de ella. Para ello será necesario:

Terminal 1: Se lanza el fichero "*.launch*" del modelo del vehículo sensorizado **Smart Elderly Car**, formado por los diferentes sensores con los que se capturan los objetos del entorno. Para ello será necesario acceder al espacio de trabajo donde se encuentra el archivo empleando las siguientes secuencias:

```
~ cd delfin_ws
~ roslaunch smart_eld_car joy_navigation.launch
```

Terminal 2: Se ejecuta el programa V-REP, donde se cargarán las diferentes escenas donde se encuentran los objetos que formarán la base de datos. Para ello, será necesario acceder a la carpeta donde se encuentra el programa V-REP y ejecutar la siguiente sentencia, de la siguiente forma:

```
~ cd VREP
~ ./vrep.sh
```

En este caso se ha creado una base de datos de un vehículo un peatón, girando cada objeto sobre su eje Z cada 5º (en caso del LIDAR) y cada 10º (en caso de las cámaras) hasta completar los 360 º. Esto se realiza seleccionando la base del objeto, y abriendo el menú de "*Rotación de objeto*":

Variando el ángulo *gamma*, se gira el objeto sobre el eje Z:

NOTA: para el caso de la cámara, se deberá emplear un terminal adicional que convierte la imagen en nube de puntos:

```
~ cd delfin_ws
~ roslaunch smart_eld_car convert_depth_to_pointcloud.launch
```

Terminal 3: En el siguiente terminal, se irán lanzando los nodos correspondientes para realizar la base de datos:

1. *Extracción del cluster:* En primer lugar, se lanza el algoritmo correspondiente para separar cada objeto de la escena y almacenarlo. Se selecciona la carpeta dentro del espacio de trabajo donde se almacenarán:

   ```
   ~ cd delfin_ws/src/delfin/src/base_datos
   ~ rosrun delfin delfin_barea_cluster_extraction_viewer
   ```

2. *Almacenamiento del cluster:* una vez extraído cada cluster, se procede a comprobar los archivos .pcd guardados, para desechar los archivos que no sean validos. Para ello, en la misma carpeta de antes, se lanza el siguiente fichero, el cuál preguntará mediante el terminal si se quiere almacenar el archivo o no:

   ```
   ~ rosrun delfin pcd_read
   ```

   En caso de que se generen dos archivos pertenecientes a un mismo cluster, se lanzará un nodo que los unirá:

   ```
   ~ rosrun delfin concatenate_pcl
   ```

3. *Escalado de la base de datos:* El último paso será escalar la base de datos. Para ello, se fijará una altura común para todos los objetos. Se lanza el siguiente fichero:

   ```
   ~ rosrun delfin scale_pointcloud
   ```

4. *Características VFH:* Creada ya la base de datos, se obtienen las características VFH de cada objeto, mediante el siguiente fichero:

   ```
   ~ rosrun delfin PCL_VFH_loop_extraction
   ```

7.2 ENTRENAMIENTO Y TEST DE LA BASE DE DATOS

Con los histogramas generados de cada archivo de la base de datos, se crean los archivos necesarios para realizar el reconocimiento de diferentes objetos dentro de una escena. Para ello, se accede a la carpeta principal del paquete donde se encuentra el archivo y se ejecuta el siguiente nodo:

```
~ cd delfin_ws/src/delfin/src
```

Se realiza el entrenamiento a la base de datos correspondiente, seleccionando al final su carpeta donde se encuentran los histogramas de la base de datos:

```
~ rosrun delfin recognition_6DOF_VFH_training base_de_datos/
```

Se generan los tres archivos explicados en el apartado 3.5.4.

Para la fase de testing, se selecciona un archivo perteneciente a la base de datos y se compara con ella de la siguiente forma:

```
~ cd delfin_ws/src/delfin/src

~  rosrun delfin recognition_6DOF_VFH_testing archivo_vfh.pcd
```

7.3 RECONOCIMIENTO DE OBJETOS

Una vez creada la base de datos y los correspondientes ficheros para los descriptores VFH, se seguirán los siguientes pasos:

Terminal 1: Se lanza el fichero "*.launch*" del modelo del vehículo sensorizado **Smart Elderly Car**:

```
~ cd delfin_ws

~ roslaunch smart_eld_car joy_navigation.launch
```

Terminal 2: Se ejecuta el programa V-REP, donde se cargarán las diferentes escenas formadas por diferentes obstáculos (detenidos o en movimiento):

```
~ cd VREP

~ ./vrep.sh
```

NOTA: para el caso de la cámara, se deberá emplear un terminal adicional que convierte la imagen en nube de puntos:

```
~ cd delfin_ws

~ roslaunch smart_eld_car convert_depth_to_pointcloud.launch
```

Terminal 3: se ejecuta el archivo "*.launch*" que genera la nube de puntos en color, de la siguiente forma para el LIDAR:

```
~ cd delfin_ws

~ roslaunch but_calibration_camera_velodyne coloring.launch
```

Para las cámaras, se emplea un archivo diferente:

```
~ cd delfin_ws

~ roslaunch but_calibration_camera_velodyne
coloring_option_2.launch
```

Terminal 4: por último, se ejecuta el nodo que separa la nube de puntos por colores y objetos, realiza un seguimiento de su trayectoria y el reconocimiento, todo ello desde el directorio donde se encuentra la carpeta con la base de datos y los archivos generados en la fase "training":

```
~ cd delfin_ws

~ rosrun delfin publish_object_vlp16_tracking % Caso Lidar

~ rosrun delfin publish_object_hdl64e_tracking % Caso cámaras
```

8 PLIEGO DE CONDICIONES

En este apartado, se describirá el hardware y software necesario para realizar este proyecto:

8.1 REQUISITOS HARDWARE:

Para este proyecto, se empleará un **Ordenador Portátil Asus F552C** con las siguientes características:

Componente	Características
Procesador	Intel Core i7-3537U CPU 2 GHz
Tarjeta Gráfica	NVIDIA GEFORCE 710 M
Sistema Operativo	Ubuntu 14.04 de 64 bits.
RAM	6 GB

Tabla 16: Requisitos Hardware del proyecto

8.2 REQUISITOS SOFTWARE:

A continuación, se exponen los requisitos software:

Software	Características/Versiones
Ubuntu	14.04
ROS	índigo
RViz	R1.11.15
V-REP Pro edu	3.4.0
Librerías	· *Point Cloud Library:* https://github.com/PointCloudLibrary/pcl · *but_velodyne* https://github.com/robofit/but_velodyne_lib https://github.com/robofit/but_velodyne · *precision_tracking* https://github.com/davheld/precision-tracking · *smart_eld_car*

Tabla 17: Requisitos Software del proyecto

9 PRESUPUESTO

En este apartado se exponen los diferentes costes para el desarrollo del proyecto, tanto materiales como profesionales.

9.1 COSTES HARDWARE Y SOFTWARE

Se exponen los costes materiales del proyecto, formados por el software y hardware empleado:

	Concepto	Coste
Hardware	Ordenador Asus F552C	700 €
Software	Ubuntu 14.04	0 €
	ROS + RViz	0 €
	V-REP	0 €
	Librerías	0 €
	Microsoft Office 365	0 €

Tabla 18: Costes materiales del proyecto

Como hardware solo se emplea el ordenador de la programación, debido a que el proyecto se realiza en un entorno de simulación.

El software empleado tiene un valor nulo, debido a que tanto Ubuntu como los programas y librerías empleados para el desarrollo del proyecto son de carácter libre.

Para Microsoft Office se emplea la licencia de estudiantes ofrecida por la Universidad de Alcalá.

9.2 COSTES PROFESIONALES

A continuación, se exponen los costes de la mano de obra necesaria para realizar el proyecto, formada por el ingeniero que se encarga del desarrollo software, y la escritura del proyecto:

Concepto	Tiempo (meses)	Coste (€/mes)	Coste total
Ingeniería	4	1350 €/mes	5400 €
Escritura	1	700 €/mes	700 €

Tabla 19: Costes de mano de obra del proyecto

9.3 COSTES TOTALES

En la siguiente tabla constan los costes totales, añadiéndoles el IVA del 21 %:

Concepto	Costes (€)
Costes materiales	700 €
Costes profesionales	6100 €
Subtotal	6800 €
IVA (21%)	1428€
TOTAL	**8228 €**

Tabla 20: Costes totales del proyecto

10 BIBLIOGRAFÍA

[1] R. B. Rusu, G. Bradski y J. H. R. Thibaux, «Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram,» de *Proceedings of the 23rd IEEE International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[2] «Acerca de Point Cloud Library,» [En línea]. Available: http://pointclouds.org/.

[3] M. Himmelsbach, T. Luettel y H. Wuensche, «Real-time Object Classification in 3D Point Clouds Using Point Feature Histograms,» de *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[4] «Grupo de robótica de la Universidad de León,» [En línea]. Available: http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors).

[5] «VLP-16 User Manual,» [En línea]. Available: http://velodynelidar.com/.

[6] A. Saxena, S. H. Chung y A. Y. Ng, «3-D Depth Reconstruction from a Single Still Image,» *International Journal of Computer Vision,* vol. 76, nº 1, pp. 53-69, Enero 2008.

[7] N. Grandón-Pastén, D. Aracena-Pizarro y C. L. Tozzi, «RECONSTRUCCIÓN DE OBJETO 3D A PARTIR DE IMÁGENES CALIBRADAS,» *Ingeniare. Revista chilena de ingeniería,* vol. 15, nº 2, pp. 158-168, 2007.

[8] «Stereo Sensor: Bumblebee XB3 1394b,» [En línea]. Available: https://www.ptgrey.com/bumblebee-xb3-1394b-stereo-vision-camera-systems-2.

[9] «Acerca del Sistema Operativo Robótico (ROS),» [En línea]. Available: http://www.ros.org/.

[10] «Acerca de RViz,» [En línea]. Available: http://wiki.ros.org/rviz.

[11] «Acerca de simulador V-REP,» [En línea]. Available: http://www.coppeliarobotics.com/.

[12] L. A. Alexandre, «3D Descriptors for Object and Category Recognition: a Comparative Evaluation,» January 2012.

[13] Y. Salih, A. Malik, D. Sidibé, M. Simsim, N. Saad y F. Meriaudeau, «COMPRESSED VFH DESCRIPTOR FOR 3D OBJECT CLASSIFICATION,» de *3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON).*, 2014.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR